Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

Thomas Winter

Online and Real-Time Dispatching Problems

22. Dezember 1999

1. Referent: Prof. Dr. Uwe T. Zimmermann
2. Referent: Priv.-Doz. Dr. Michael L. Dowling
eingereicht am: 14. Oktober 1999

Zusammenfassung

In der vorliegenden Dissertation werden zwei miteinander verwandte Dispositionsprobleme in Betriebshöfen des ÖPNV und in Übersee-Containerterminals untersucht. Beide Probleme sind in der Praxis unter Echtzeitbedingungen zu lösen.

Beim Betriebshofdispositionsproblem sind den Fahrzeugen (Straßenbahnen), welche in den Betriebshof einfahren, sowohl Stellplätze als auch die als nächstes zu bedienenden Umläufe zuzuweisen. Die Modellierung dieses Problems erfolgt mittels einer Klassifizierung der Fahrzeuge in Typenklassen und mittels Stacks, die zur Modellierung der Abstellgleise (Stumpfgleise) für Straßenbahnen dienen. Es werden zudem Zusammenhänge und Anwendbarkeit des vorgestellten Modellansatzes für Busdepots und Betriebshöfe mit Durchfahrtgleisen diskutiert.

Als Optimierungskriterium werden zwei zu minimierende Zielfunktionen betrachtet: zum einen die Anzahl von Fahrzeugpaaren, die zu rangieren sind, und zum anderen die Anzahl von Zuweisungen von Fahrzeugen einer Typenklasse zu Umläufen, welche jedoch eine andere Typenklasse verlangen. Im ersten Fall sind nur typenreine Zuweisungen von Fahrzeugen zu Umläufen erlaubt, im zweiten Fall nur rangierfreie Zuweisungen. Beide Problemstellungen erweisen sich als $\mathcal{NP}$-schwer. Zur Minimierung des Rangieraufwandes wird ein binäres quadratisches Programm vorgestellt, welches sich aus zwei binären quadratischen Zuordnungsproblemen zusammensetzt, die über Typenbedingungen gekoppelt sind. Für dieses Programm werden Linearisierungen und untere Schranken mittels LP-Relaxation betrachtet. Für das Problem der Minimierung der Typenverletzungen werden ein binäres lineares Programm angegeben, sowie obere und untere Schranken für den Zielfunktionswert bewiesen.

Als Spezialfall wird das Betriebshofdispositionsproblem für eine bereits gegebene Zuweisung von Fahrzeugen zu Stellplätzen betrachtet. Das resultierende Teilproblem erweist sich für beide Zielfunktionen als $\mathcal{NP}$-schwer. Ausgehend von den Ergebnissen und dem Modell für das allgemeine Dispositionsproblem werden ganzzahlige Programme und ein dynamischer Programmierungsansatz vorgestellt sowie polynomial lösbare Spezialfälle betrachtet.

Sowohl für das allgemeine Betriebshofdispositionsproblem als auch für die Spezialfälle werden die Ergebnisse für exakte und heuristische Verfahren anhand von zufällig erzeugten Daten und Praxisdaten verglichen. Es zeigt sich, daß selbst bei engen Echtzeitanforderungen gute, zum Teil optimale Ergebnisse erzielt werden können.

Eine kompetitive Analyse erfolgt für die online Versionen der betrachteten Optimierungsprobleme. Hierbei wird der Begriff der Kompetitivität auf die, für das Betriebshofdispositionsproblem häufig auftretende Situation einer optimalen Lösung vom Wert Null erweitert. Zu diesem Zweck führen wir den Begriff der (c, d)-Kompetitivität ein. Wir beweisen obere und untere Schranken für die Kompetitivität beliebiger deterministischer und randomisierter online Algorith-

men und untersuchen das Verhalten von Greedy-Algorithmen für die Spezialfälle. Für das Echtzeitdispositionsproblem in Betriebshöfen stellen wir basierend auf den ganzzahligen Optimierungsmodellen echtzeitfähige Algorithmen vor und bewerten ihr Verhalten anhand von Praxisdaten.

Abschließend diskutieren wir verwandte Stacksortierungsprobleme und stellen den Zusammenhang des Betriebshofdispositionsproblems zu Stauplanungsproblemen auf Containerschiffen und in Containerterminals dar. Wir präsentieren einen neuen, integrierten Echtzeitplanungsansatz zur kombinierten Stau-, Lade- und Transportplanung in Übersee-Containerterminals. Dieser Ansatz wird anhand von Praxisdaten getestet.

Acknowledgements

The tram dispatch problem presented in this PhD thesis was motivated by a joint project with DaimlerChrysler Rail Systems (Signal) GmbH, Braunschweig, former "Ingenieurgesellschaft für Verkehrsplanung und Verkehrssicherung" (IVV) GmbH, Braunschweig. The research work on this project was partly supported by the German Research Foundation (DFG). The real-world data that form a significant basis for the results obtained in this thesis were supplied by the following German transport companies: BVAG (Braunschweig), HAVAG (Halle), VBK (Karlsruhe), MVB (Magdeburg), and WVG (Wolfsburg). Moreover, many helpful suggestions were made by IVU Traffic Technologies AG, Berlin. The container dispatch problem was considered in a research project with the Hamburger Hafen- und Lagerhaus AG (HHLA). This project has been partly supported by DFG. I would like to express my appreciation of the support of all these companies and institutions. In particular, I would like to thank Uwe Strubbe (IVU) and Dirk Steenken (HHLA) for many helpful remarks and discussions.

My sincere gratitude goes to Professor Uwe Zimmermann for supervising this thesis, for his constant guidance, his support and encouragement. I appreciate to benefit from his long and rich experience with combinatorial optimization problems and integer programming.

I would like to thank all the members of the Department of Mathematical Optimization for the pleasant time at work and the fruitful "discussions" we had during the coffee breaks. In particular, I am grateful to Michael Bussieck and Hannes Scheel for their work and their suggestions while working on the problem of scheduling trams in the morning (STIM) and to Jan van der Veen who implemented the first version of the RTS heuristic.

I am deeply indebted to Nicola Kamin who helped me in many ways. I enjoyed the discussions of the different dispatching problems and of several ideas I had. I thank her for many helpful remarks and for proof-reading of this thesis.

Finally, I would like to thank my family and friends for their love, support, and understanding.

Contents

Chapter 1

Introduction

Dispatching problems play an important role in the planning process of transport companies and logistic centers. The way in which the available transport vehicles and the stored goods are stored has often direct influences on the productivity and quality of the complete planning process. In the most cases, the arising dispatching problems are too complex so that it seems that it is nearly impossible to compute optimal solutions. Usually, "good" or "acceptable" solutions are determined manually by dispatchers where the quality of the solution depends in the dispatcher's experience. With the integration of computers in the decision process, combinatorial optimization methods have found application, for instance, in decision support systems that help the dispatcher in finding good decisions.

In the investigation of real-world optimization problems, the notions "online" and "real-time" have become more and more of interest within the last years. Due to the major improvement of computer power which is accompanied with the increasing quality and availability of computerized data, it is possible to consider and to evaluate online and real-time influences on complex real-world systems. At the same time, new sophisticated combinatorial optimization methods have been developed and implemented. Online problems arising in real-world logistic systems have been examined for instance by Ascheuer and Kamin [Asc95, Kam98]. Besides this practical aspects, theoretical investigations in online problems, so-called *competitive analysis*, lead to a first performance measure for online algorithms. For a worst case instance, the solution value achieved by an online algorithm is compared with the solution value that an optimal algorithm yields if it is provided with complete information about the problem.

In this thesis, we examine online and real-time dispatching problems which arise in local transport and container logistics. We start with considering the dispatch of trams in depots of local transport companies where the trams are stored before they serve the next round trips. Due to delays and other external effects, the actual arrival order of trams differ from the order implied by the daily schedule. The actual sequence of arriving trams is not known in advance so that we have to deal with this incomplete information. However, on arrival the trams

have to be assigned to a location in the depot. This decision has to be made online without any knowledge of future events. On arrival, the tram shall also be assigned to a round trip of the next schedule period. This choice of the next round trip may influence the decision where to store the tram.

The time available for computing a good choice for the location and the next round trip is limited. The reason for this is that, due to the lack of space, the tram cannot wait a couple of minutes at the depot entrance. Hence, a decision has to be computed within a given time bound depending on the gap between two arrivals.

Consequently, a decision has to be found in real-time, which means within a given time bound and while the dispatch process continues (cf. [But97]). Usually, this means also that the decision has to be made fast, for instance, within less than one or two minutes. Due to the complexity of the whole dispatch process, a complete reoptimization of the system is impossible. Therefore, based on an analysis of the problem's structure, sophisticated methods have to be designed and evaluated that guarantee a good decision within the required time.

The second problem field briefly considered in this thesis arises in maritime container terminals. After discharging the import containers, all export containers have to be transported to the container ship, to be loaded onto the ship, and stored at the bay positions of the ship. The corresponding stowage and loading plan is usually based on restrictions provided by the shipping company. Since a large number of export containers arrive at the terminal after the loading process has already begun, we have to deal with incomplete information about the containers. The dispatcher has to react on delays and changes in the stowage plan by updating the loading and transport sequences. These changes have to be carried out in real-time within tight time bounds while the loading and transport of containers continues.

Outline of the Thesis

In Chapter 2, we present the different dispatching problems considered in this thesis. We introduce briefly the dispatch problems for trams and busses in depots of local transport companies. In these problems, arriving trams have to be assigned to locations in the depot and to the round trips given by the schedule. We point out common problems in bus and tram dispatch. Based on a definition of types into which the transport vehicles are grouped, we describe a mathematical model and different objectives for these dispatching problems. Moreover, we give an introduction to online and real-time dispatch problems at storage yards. As related problems, we describe a dispatch problem of locomotives at terminus stations in rail traffic and some problems arising in maritime container terminals.

In Chapter 3, we examine the computational complexity of the different dispatching problems at storage yards for trams, i.e., the tram dispatch problem TDP where the number of shunting movements is to be minimized and the type

mismatch problem TMP where shunting of trams must be avoided completely. We show that both problems are $\mathcal{NP}$-hard by giving a reduction from three dimensional matching to the corresponding decision problems of TDP and TMP. We show that the versions of TDP and TMP restricted to the departure part remain $\mathcal{NP}$-hard. Additionally, some polynomially solvable cases are presented.

In Chapter 4, we present mathematical programming models and algorithms for the tram dispatch problem and the type mismatch problem. For the tram dispatch problem, we give a binary quadratic program which is a combination of two quadratic assignment problems. We examine the situation in which shunting is required and present a linearized mixed integer program. Two heuristics are introduced and evaluated. For the type mismatch problem, we give a binary linear program and discuss the relation between the tram dispatch and the type mismatch problem. We present computational results for the exact and heuristic methods applied to some real-world and random data.

Chapter 5 is devoted to the dispatch problems at departure. Based on the results obtained in Chapter 4, we present exact and heuristic methods for minimizing the amount of shunting and for the type mismatch problem. In particular, we discuss dynamic programming approaches for the departure dispatch problems and the application of heuristic methods. Computational results are presented for real-world and random data.

In Chapter 6, we give an introduction to competitive analysis and related performance measures for online algorithms. We motivate the techniques for evaluating the performance of online algorithms by giving a survey on some results for classical online problems as well as on results for some online versions of classical combinatorial optimization problems.

In Chapter 7, we extend the notion of competitiveness for the online tram dispatch problems, i.e., for the situation where the considered instances are solvable with zero cost. We distinguish between instances with optimal value of zero and such instances that require non-zero cost. For this situation, we coin the notion of (c, d)-competitiveness. We present some lower and upper bounds on the competitiveness of arbitrary online algorithms and some particular online algorithms for the arrival-departure dispatch problems and the departure dispatch problems.

Chapter 8 is concerned with real-time algorithms for the tram dispatch problem. We start with a description of the real-time tram dispatch process in storage yards. We discuss the application of the presented real-time algorithms on the basis of computational results for real-world and random data and for two different real-time scenarios.

In Chapter 9, we show some connections between the tram dispatch problem and sorting problems for stacks. We introduce a classical permutation sorting problem and some recently proposed problems arising in literature. Moreover, we investigate some connections between tram dispatch, train dispatch, and container logistics. We show how the results for the container stowage problem may be applied to tram dispatch.

In Chapter 10, we briefly discuss several dispatching problems at maritime container terminals. We propose an integrated approach for the combination of ship and transport planning and present computational results for a real-time algorithm.

Chapter 11 is concerned with concluding remarks on the results obtained in this thesis and on the applicability of the derived real-time methods.

Chapter 2

Dispatching Problems

In transport and logistics, several dispatching tasks have to be solved in order to guarantee high quality of service at reasonable cost. In the last decade, computational methods for improving the dispatch process became more and more important in the daily planning work of transport companies and logistic centers. One reason for this effect was the major improvement of computer power resulting in a reduction of computation time for solving complex dispatching problems. Another reason were new sophisticated mathematical methods which are indispensable in order to solve real-world optimization problems.

With the integration of computers in the planning process, a huge amount of (computerized) data became available for a detailed process analysis. This data enabled the use of discrete optimization methods in order to compute better and more global solutions.

2.1 Dispatching in Public Transport

In public rail transport and local transport, the circulation of rolling stock and the daily schedule (usually a seasonal and periodical schedule) are determined in a hierarchical planning process [BWZ97]. The round trips of the daily schedule are formed, transport vehicles of suitable type are assigned to them, and for each round trip a schedule is fixed. During this planning process, the crews are assigned to the round trips of the transport vehicles. In each step of this planning process, operational constraints given by the infrastructure of the company as well as constraints given by law and working agreements must be taken into consideration. The solution obtained in each step has to be consistent with respect to every solution of the other subproblems. Due to the complexity of the problem, even for small transport companies it is impossible to determine a global optimal solution for the complete dispatching process within the required time bounds.

The planning process in public transport consists of several steps (see Fig-

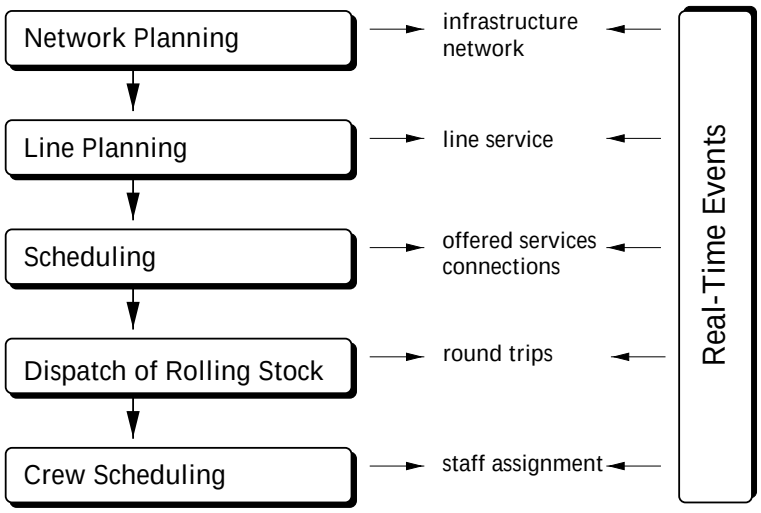

Figure 2.1.1: The hierarchical planning process in public transport.

ure 2.1.1). The first step is the network planning process. Usually, the corresponding transport network is historically grown. From time to time, the transport companies decide to build new stations or to extend or reduce service to certain locations. Decisions made within this step are long-term decisions, since they often require high costs and construction work. This process is strongly influenced by political reasons. The infrastructure of the network developed in this step forms the basis for all further planning steps.

In the second step, the companies decide which services between different terminal stations they will offer. Such a service, called line, is a route in the network and connects two terminal stations. Usually, a line is served periodically within a fixed time interval. The frequency of the lines should be chosen in order to satisfy the passenger demand at a low cost level. Different optimization approaches are presented and developed for example by Bussieck [Bus98].

The line planning process is followed by the third step: the computation of the schedule. Based on the line plan and the frequencies of the lines, the arrival and departure times are calculated for each station and each connection of the line. Several side constraints, for instance safety regulations, have to be taken into account. Usually, the schedule is determined by a team of experienced planners supported by computer systems.

In the fourth and the fifth step of the planning process, the available rolling stocks and the available personnel are assigned to the itineraries and the duties. As a result, the round trips are computed and the personnel is assigned to them. Depending on the length of a round trip, it may be partitioned into several parts to which different personnel has to be assigned.

All these steps are strongly connected. Computing an optimal solution in one

step may restrict the possibilities in the next step such that the global solution may be not satisfactory. Therefore, the calculation in each step should take care of the forthcoming objectives in order to avoid a complete recalculation. Usually, this approach is based on the knowledge of high-experienced dispatchers.

2.1.1 Dispatching in Storage Yards

During the planning process in local transport companies, one usually ignores how the dispatch has to be organized at the depots. At each depot, the planned (virtual) assignment of vehicles to round trips has to be done. For instance, a vehicle of suitable type has to be assigned to each round trip. This assignment process often suffers from the lack of space inside of the depots. In many depots, it is impossible to assign the transport vehicles to round trips without rearranging some vehicles. When the vehicles arrive at the depot after having served their scheduled round trips, another problem arises. The arriving vehicles have to be stored inside the depot. The choice of this standing position should take into account the dispatch plan of the next schedule period. In some cases, the dispatching process of arriving and departing vehicles interact.

In this thesis, we focus on dispatching problems in local transport. We distinguish between dispatching in bus depots and dispatching in storage yards for trams. Usually, bus depots and storage yards for trams are organized differently. As reality shows, each depot and storage yard is organized in a specific way depending on the local dispatching possibilities. We will focus on common problems and strategies which can be applied to bus depots as well as to storage yards for trams. Before we discuss the dispatching problems from the mathematical point of view, we give a detailed description of the planning process in storage yards. Then, we discuss the problem for different storage yards for trams and point out similarities between bus depots and such storage yards for trams.

2.1.2 Storage Yards for Trams

Storage yards for trams can be divided into three classes of depots depending on the track configuration inside the depot (cf. Figure 2.1.2 and Figure 2.1.3):

1. depots with dead-end sidings
2. depots with loop-lines
3. depots with dead-end sidings and loop lines

As we have stated in the last section, the concrete dispatch process differs from depot to depot. We give some examples for some German storage yards and depots. We start with a description of the Braunschweig storage yard for trams. Next, we compare the situation in Braunschweig to the situation at the Magdeburg tram depot “Betriebshof Nord” and give some remarks to the situation at the storage yard “Karlsruhe-Ost”.

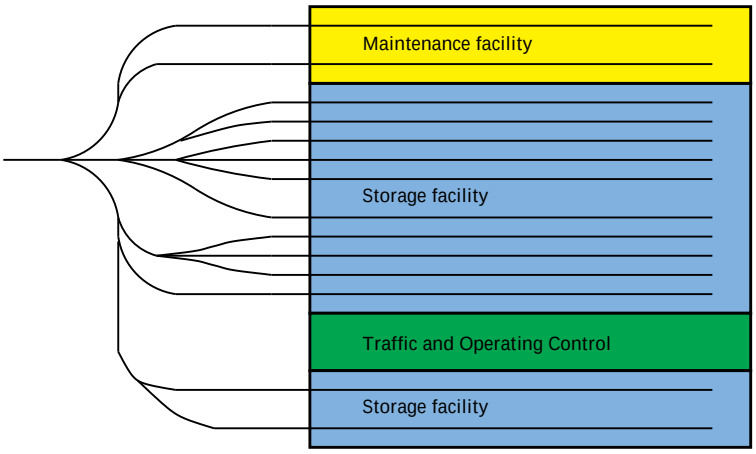

Figure 2.1.2: A depot consisting of dead-end sidings.

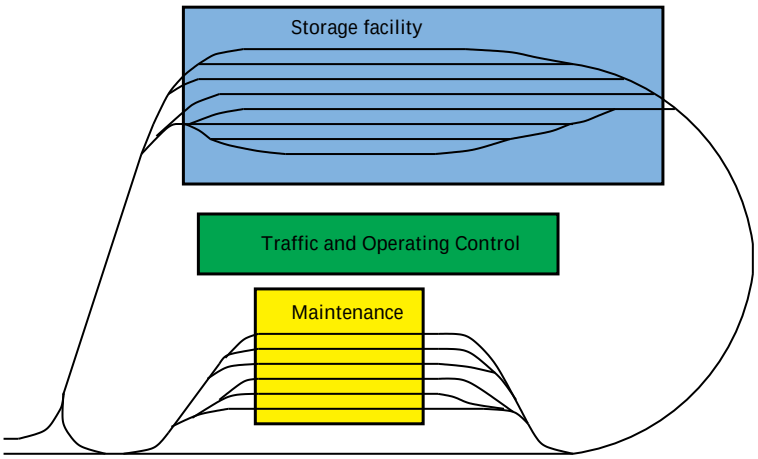

Figure 2.1.3: The Magdeburg storage yard: a depot with loop lines.

2.1.3 The Braunschweig Storage Yard

The Braunschweig local transport company "Braunschweiger Verkehrs-AG" (BVAG) operates one storage yard for trams. In this storage yard, up to 48 trams are stored. The BVAG owns low floor trams of type GT8S (1995), trams of type "Braunschweig" (LHB, 1980/81), and cars of type "Mannheim" of three different construction years (1973/1975/1977) [LRT96, AGB96]. The trams can be divided into seven classes of trams. The first class consists of the low floor trams, the second and the third class contain all "Braunschweig" type trams where the latter class contains tram-sets (car/trailer). The remaining classes consists of the "Mannheim" type trams, where the 1977-type trams are divided into two subclasses (tram-sets/cars).

- class 1: low floor trams of car type GT8S
- class 2: trams of car type "Braunschweig" (LHB, 1980/81)
- class 2: tram-sets of car type "Braunschweig" (LHB, 1980/81)
- class 3-5: trams of car type "Mannheim' (1973/1975/1977)
- class 7: tram-sets of car type "Mannheim' (1977)

Trams are of the same class if they are of the same construction year and if they are identically equipped. To some trams, a trailer can be coupled. A class consists of a homogeneous tram fleet. The construction year and the trams' equipment is not the only difference between these trams. For instance, the trams also differ by their length, their passenger capacity, and their qualification for the transport of handicapped people.

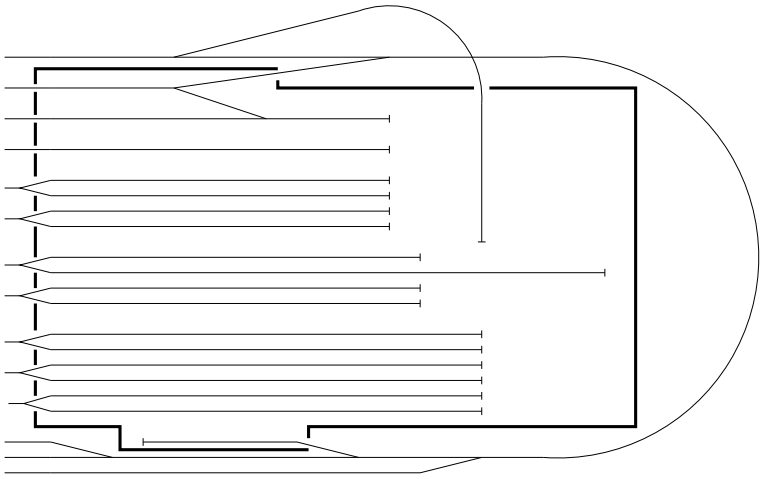

Figure 2.1.4: The layout of car barn at the Braunschweig storage yard for trams.

In the Braunschweig storage yard, only one track is leading into the depot. On this track, all trams must enter and leave the depot. The track splits into 17 different dead-end sidings that lead into the car barn of the storage yard (cf. Figure 2.1.4). The sidings are not of the same length. Inside the car barn, there are some sidings reserved for the maintenance of the trams. Outside the building, there is an additional holding yard consisting of a siding besides the car barn and some short sidings near the depot entrance. On these sidings, all trams have to be stored. There are several restrictions for assigning the trams to the sidings. For security reasons, certain escape routes have to be kept free. The configuration of guideways and switches implies that not every siding is reachable for the low floor trams.

Every five days, the trams have to go to the maintenance facility for their scheduled preventive maintenance. The maintenance is done on the first siding such that usually shunting of trams is necessary. Each shunting of trams requires approximately five minutes [BVA96].

Due to the track and switch configuration of the Braunschweig tramway net (see Figure 2.1.5), not all lines can be served by every arbitrary tram. Some lines must be served by a certain class of trams. For instance, the tram line 2 cannot be served by low floor trams but should be served by the 1981-type trams, since this line is frequently used by old people. The BVAG also cares for the advertising livery of the trams. Some advertising customers of the BVAG expect that the tram carrying their slogan serves certain (popular) lines following tracks through the city center. The treatment of such particular restrictions differs from company to company and from town to town. Some transport companies refuse to care for advertising when choosing trams for round trips.

In Braunschweig, the tram lines meet in the city center where the lines share the same tracks. Here, the headway of trams is only a few minutes. Only minor delays may result in a different ordering in which the trams serve the stations in the center. Moreover, most of the pull-in trips start from one of these stations. Hence, a change in the order of service will usually influence the arrival order of trams at the depot.

2.1.4 The Magdeburg Storage Yard "Betriebshof Nord"

At the depot "Betriebshof Nord", the Magdeburg transport company "Magdeburger Verkehrsbetriebe" (MVB) operates 25 low floor trams and 61 Tatra cars of type T4D to which 32 carriages could be coupled [MVB96, AGB96]. These trams are stored on eight sidings of length between 400 and 500 meters. Usually, the MVB put on three classes of trams and tram-sets: the low floor tram (NGT), two coupled Tatra cars (T4D), and two coupled Tatra cars with a carriage (B4D).

The "Betriebshof Nord" is a depot with loop lines (cf. Figure 2.1.3). After having served their round trips, the trams arrive at the storage yard's entrance. After vehicle identification, the dispatcher informs the driver of the arriving tram

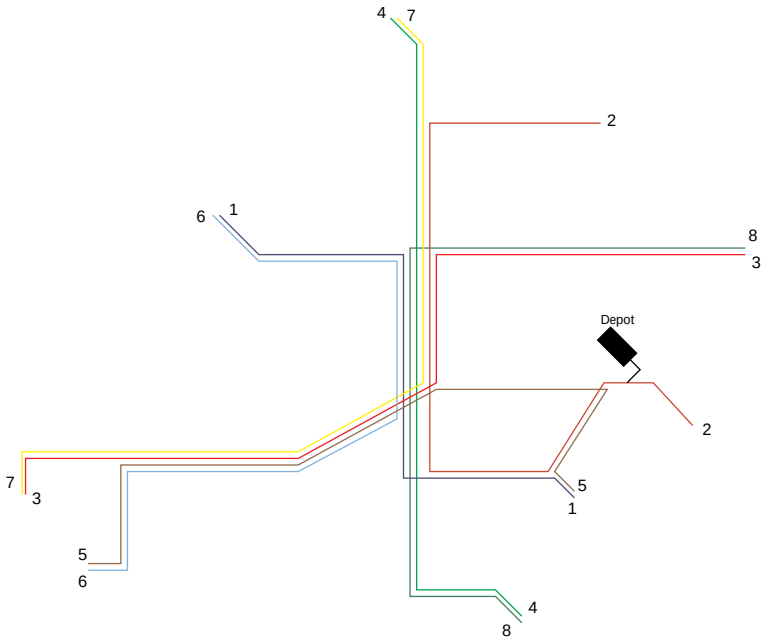

Figure 2.1.5: The Braunschweig tramway line plan.

whether the tram should be stored directly at one of the siding positions or whether it should go directly to be serviced at the maintenance facility. The way to the siding position follows a loop line such that the siding is reached from the back end. Sometimes a tram is first stored on one of the sidings before maintenance. In this case, shunting of trams may be necessary in order to move this tram to the maintenance facility. In general, at "Betriebshof Nord" shunting of trams means that these trams have to follow the complete loop line before reaching the new standing position at the end of a siding. Such an operation takes time and requires a ride of approximately three kilometers. Only exceptionally, it is possible for some trams to enter the sidings from both sides.

2.1.5 The Storage Yard "Karlsruhe-Ost"

The layout of the storage yard of the Karlsruhe local transport company VBK is illustrated in Figure 2.1.5. It consists of dead-end sidings as well as of sidings that can be accessed from the back-end. The trams leave the depot on the western side. We observe that trams stored on dead-end sidings in car barn "Halle 3" can only leave the depot on tracks leading to the South. This leads to some restrictions concerning the assignment of trams (serving particular round trips) to sidings in this car barn. Additionally, not every track can be used by every tram because of the switch configuration. This also restricts the possibilities of assigning the trams to the siding positions.

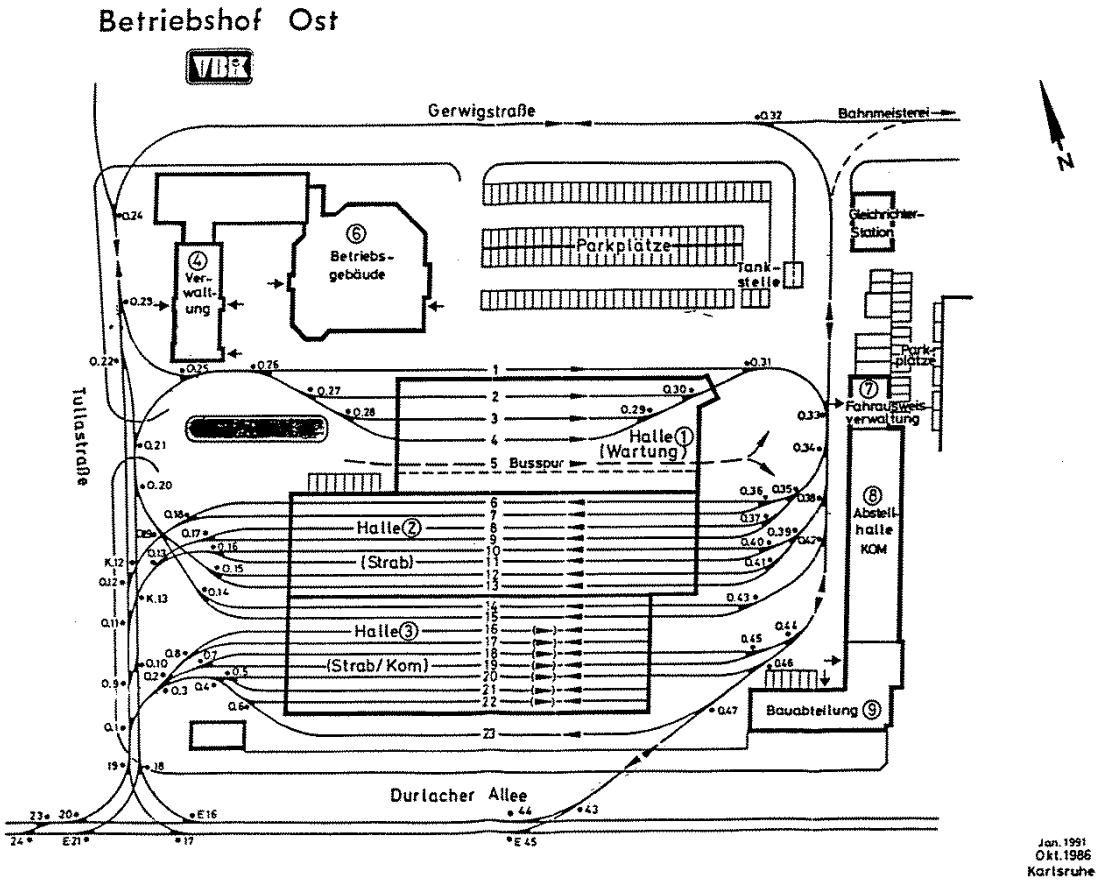

Figure 2.1.5: The layout of the storage yard “Karlsruhe-Ost”.

2.1.6 Bus Depots

Bus depots often allow a more flexible dispatch of the vehicles to the depot positions. In some cases, the holding yard offers enough space to allow the transport company to store the busses one beside the other. In such depots, shunting of busses is not necessary. In other kinds of depots, due to the lack of space or due to the layout of the holding yard, the busses are forced to stand one behind the other. In this case, it may be necessary that busses are shunted in order to let them leave the depot in a specified order. Additionally, some depots consist of separate areas which are differently organized.

In this thesis, we concentrate on the type of bus depots in which the busses are stored in lanes one behind the other. In such a depot, the lanes and the distance between two busses may offer enough space to enable a bus to pass the ones standing in front of it. Except of these possibilities, such bus depots can be handled in the same way as storage yards for trams.

An example of such a depot is the depot of the Wolfsburger Verkehrs-GmbH (WVG). At the WVG depot [WVG96], 65 busses of four types are stored: standard busses, standard articulated busses, low floor busses, and articulated low floor busses. The set of busses of one type is divided into two classes depending on the round trips that they are allowed to serve. The round trips are divided into two groups. The first group consists of usual round trips serving lines through the city of Wolfsburg. The second group contains the round trips serving the commuter transport to the Wolfsburg factory of Volkswagen (VW). Usually, the WVG does not strictly distinguish between standard and low floor busses.

2.2 The Concept of Types

In the following, we focus on storage yards for trams. A transport company operates several trams of the same class. These trams differ only marginally and hence all of them can be assigned to a subset of departures. Additionally, a round trip may be served by a variety of tram classes. Usually, there are some preferences which class of trams should serve a certain round trip. These preferences are motivated for instance by the tram's equipment and their capacity. It also depends on the track configuration whether each tram is allowed to serve every round trip. These operational constraints give us some first restrictions on the set of feasible assignments of trams to round trips.

Besides these operational constraints, the trams have to go to be serviced in fixed time intervals. If the maintenance does not require the tram to be in the depot for the whole day, the tram can be assigned to a short round trip after (or before) the maintenance. Such round trips usually occur in the rush hours in the early morning and in the evening.

Some transport companies also try to keep the mileage for trams of the same

class on the same level. One possibility to achieve such a balanced level is to subdivide the class of trams into some subclasses where the trams of the subclasses are only allowed to serve a round trip of specified length. This division into subclasses may be changed in some period, for example, daily.

In order to deal with this changing situation, we define the notion **type** for each tram and each round trip. This notion is an extension of the tram's car type. The usual tram type is defined by the type of the car. We extend the notion of type in the following way. We partition the trams into groups which usually correspond to the classes introduced above. Each group is identified by a type of tram. However, this partition need not necessarily be equal to this division into classes or fit to the partition implied by the car types. For each tram and each round trip, an abstract tram type is chosen. Additionally, for each such type, the number of trams of this type must be identical to the number of round trips of the same type.

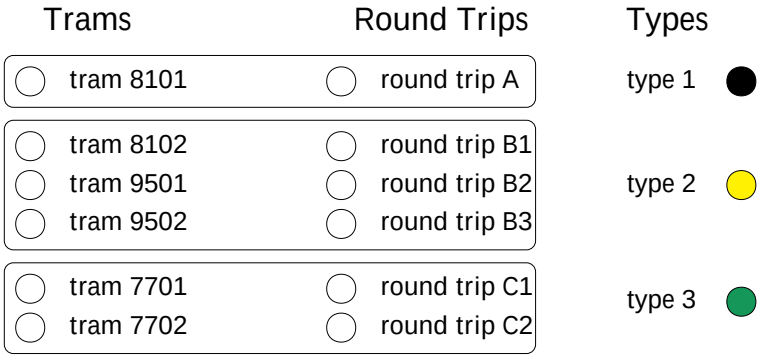

Figure 2.2.6: An example for the definition of types.

In Figure 2.2.6, we give an example for this redefinition of types. The first two digits of the tram's number indicate the construction year of the tram and hence the tram's car type. The third and fourth digit correspond to internal number of the tram. The different kinds of round trips are denoted by the capitals A, B, and C and an internal number. We group the trams in the following way: The first group consists only of tram (8101) that should be assigned to round trip A. We introduce the type 1 for this tram. The second group contains three trams (8102, 9501, and 9502). We define the second type (type 2) for this group of trams that should serve the round trips (B1, B2, and B3). Finally, the trams 7701 and 7702 form the third group which corresponds to type 3. These trams should serve the round trips C1 and C2. We represent the trams and the round trips by circles that are filled with colors corresponding to the respective types.

This definition of types allows us to model the possible assignment of trams to round trips in a more flexible way. The groups of trams of unique type can be redefined whenever the situation changes. For instance, we can repeat the

partition into types if necessary day by day based on different constraints.

Throughout the following sections, we assume that the dispatcher has fixed a type for each tram. Additionally, we assume that the dispatcher has assigned to each round trip a certain type. This means that a tram of this type should serve the round trip. This type-fixing-procedure, i.e., the division of trams into types and the assignment of types to round trips, has to be done in such a way that it is possible to find an assignment of trams to round trips where each tram is assigned to a round trip both having the same type.

2.3 A Mathematical Formulation

As shown in Figure 2.1.2 and Figure 2.1.3, the depots consist of sidings that can be accessed either from the back-end or from the front-end. For depots consisting of dead-end sidings, all the trams enter the siding from the front side and leave it on the same side. For depots of the second type (cf. Figure 2.1.3), the trams usually follow the tracks and enter the sidings from the back-end and leave it at the front-end. Only exceptionally, it is possible that trams leave and enter such drive-through sidings on the same side. On the one hand, this depends on the tram's car type, i.e., if it is possible for the driver to drive in both directions. On the other hand, such moves require additional personal staff and time for the resulting shunting trips. Starting for the next round trip, the trams always leave the sidings at the front-end.

All these kinds of sidings can be mathematically modeled by stacks, queues, or, in the last case, by dequeues. From a mathematical point of view, stacks and queues differ only marginally. A stack is handled using the **last-in-first-out (LIFO) principle** and a queue is accessed using the **first-in-first-out (FIFO) principle** (cf. Figure 2.3.7). A more detailed survey on data structures like stacks and queues can be found in [Knu68].

Remark 2.3.1: Through-out the whole thesis, we consider the situation that the depot consists of stacks only.

All the introduced methods work for queues as well. We will point out how we can easily adapt the methods for queues. We divide the stacks into several positions depending on the length of the stacks. We ignore the concrete tram length. This is motivated by the dispatch sheet used by the BVAG dispatcher (see Figure 2.3.8) and by the personal communication with the WVG dispatcher [WVG96]. For reasons of security, the escape routes through the whole barn have to be kept free such that there is a straight path from one side of the building to the other side. In Chapter 4, we point out how different tram lengths can be taken into account within our approach.

Figure 2.3.8 shows the slightly simplified dispatch sheet of the BVAG dispatcher. The sidings $1a$, 18, and 19 are outside the car barn. Sidings 18 and 19

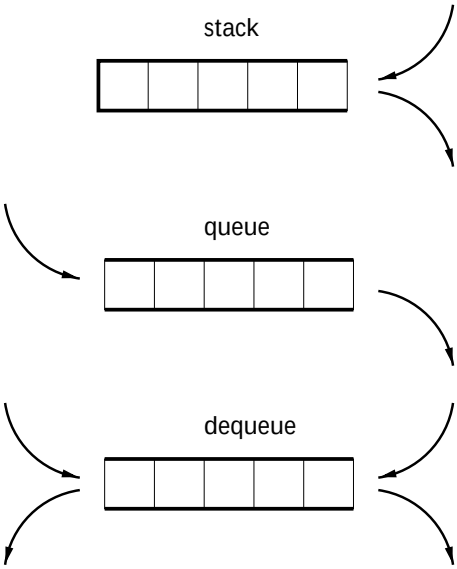

Figure 2.3.7: Stacks and queues.

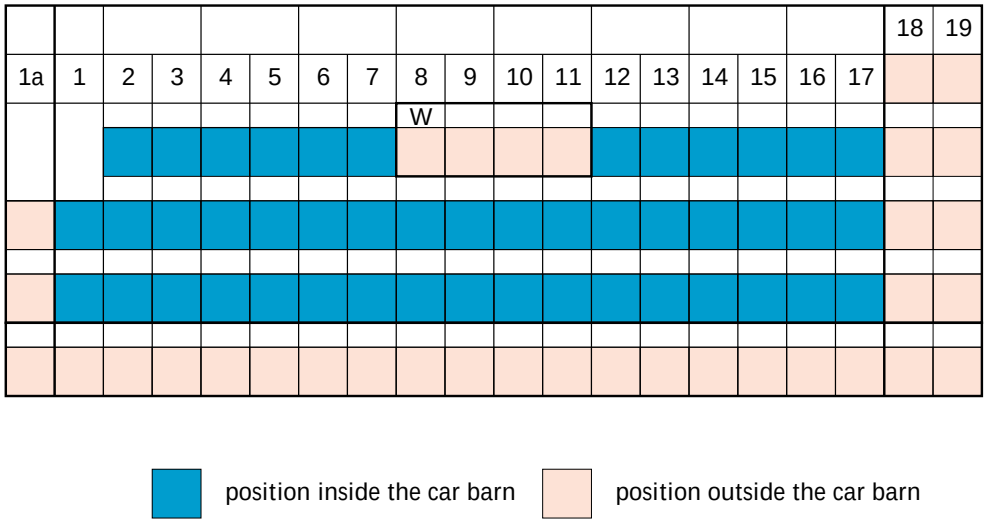

Figure 2.3.8: Dispatch sheet used by the BVAG dispatcher.

are longer than the other sidings resulting in a capacity of five trams. The letter W in the center of sheet marks those positions outside the car barn which are located next to the depot entrance.

Arrival of Trams

First, we consider the arrival of trams at the depot. For the sake of simplicity, we consider the empty parts of each stack and assume that the depot is empty before the first arrival. We will go into details in Section 4 where we will consider the more general case.

We assume that we are given a set $\mathcal{A} = \{a_1, \ldots, a_N\}$ of arriving trams. The index i of tram a_i denotes the position of a_i in the arrival sequence. For instance, tram a_1 arrives as the first, tram a_2 as the second and so on. An example for such an instance is given in Figure 2.3.9.

Each tram has to be assigned to a stack position. The amount of shunting necessary for this assignment depends on the order in which the trams are assigned to the stack positions. Whenever shunting is necessary, two trams are involved. The first tram is assigned to some stack position. When a second tram arriving earlier is assigned to a position located deeper in the stack, then the first tram has to be shunted in order to assign the second tram to its position. We observe that shunting trams may be necessary if the trams are assigned to the same stack in an order that differs from the order in which the trams arrive at the depot. An example of such a situation is given in Figure 2.3.9.

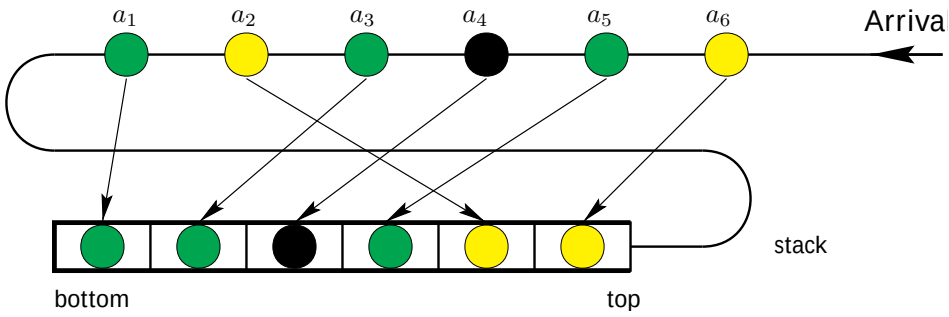

Figure 2.3.9: A model for the tram dispatch problem: the arrival. The trams a_3, a_4, and a_5 have to be shunted with a_2 on arrival.

Departure of Trams

To each round trip of the daily schedule, we associate a departure. The times at which the corresponding pull-out trips start give us the departure times. A sequence of departures is denoted by $\mathcal{D} = \{d_1, d_2, \ldots, d_M\}$ where d_1 denotes the

first and d_M denotes the last departure. As introduced in Section 2.2, for each round trip and hence for each departure we have chosen a type of tram that has to serve the round trip.

When we assign trams standing in the stacks to the departures corresponding to the forthcoming round trips, we are faced with a situation similar to the situation for the arrivals. Shunting trams at departure may be necessary, if the trams are assigned to departures in an order that differs from the order implied by the stack positions. Once again, whenever shunting is necessary, two trams are involved. We assume that a tram at some stack position is assigned to a departure d_k. Shunting is necessary if either another tram at a deeper location in the same stack is assigned to an earlier departure or a tram stored on-top is assigned to a later departure. An example for such a situation is given in Figure 2.3.10.

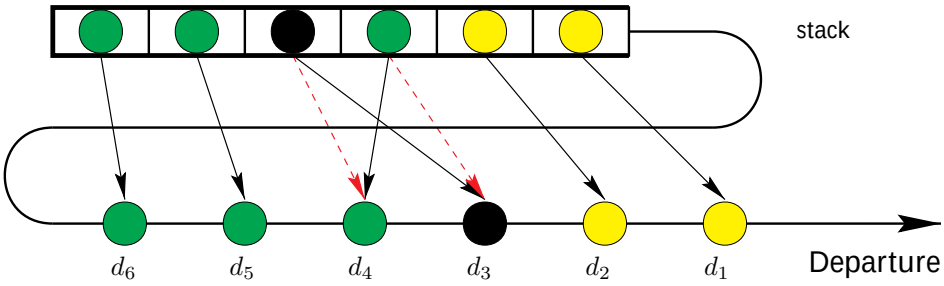

Figure 2.3.10: A model for the tram dispatch problem: the departure. The trams assigned to departure d_3 and d_4 have to be shunted at departure. If shunting is forbidden, the corresponding round trips are served by trams of non-matching type.

In the case that we want to avoid shunting at departure, for instance because of short time intervals between two consecutive departures, we possibly cannot choose a tram of matching type for each departure. Hence, we have to allow some type mismatches when assigning trams to departures (cf. Figure 2.3.10).

Arrival and Departure

A model for a combined dispatch of trams to stack positions and to departures is obtained by combining the models for the arrival and departure part in the following way (cf. Figure 2.3.11).

Each tram is first assigned to a stack position and secondly to a departure. Hence, we define an assignment of arrivals to positions and an assignment of departures to positions. The image sets of both assignments have to be identical. Shunting of trams occurs on arrival or at departure (or in both situations). Addi-

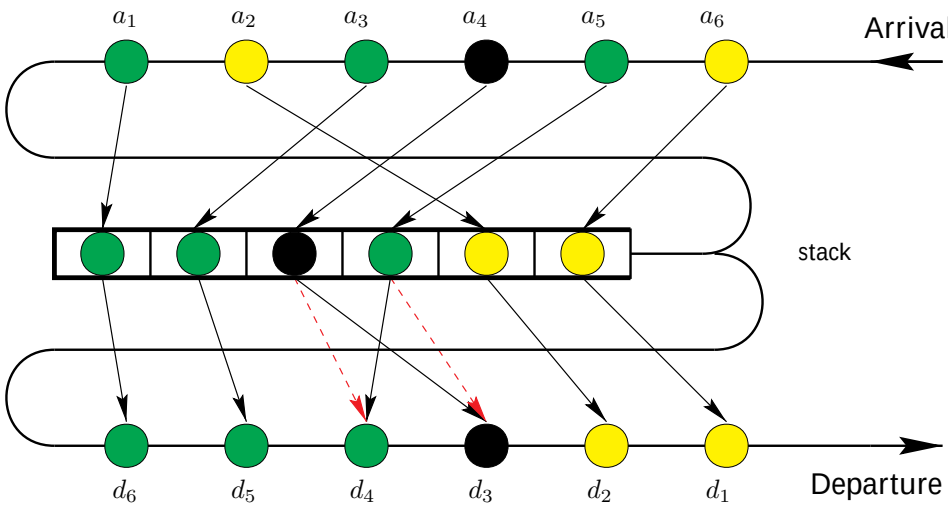

Figure 2.3.11: A model for the tram dispatch problem: Combination of arrivals and departures. The trams a_3, a_4, and a_5 have to be shunted with a_2 on arrival. Tram a_3 and tram a_4 have to be shunted at departure.

tionally, shunting only depends on the way in which the arrival and the departure sequence is assigned to the stack positions. The only connection between the arriving trams and the departures is modeled by the types that are required by the departures. In the case that we forbid to assign a tram of non-matching type to a departure, we have to force that the assignment of trams to positions and the assignment of departures to positions satisfies the property that the respective types assigned to the positions match for each position. Otherwise, we allow the assignment trams to departures that require a different type. In Chapter 4, we will go into more details.

Figure 2.3.11 illustrates an example of a depot consisting of one stack only. In this situation, all trams arriving at the depot are stored in this stack before they leave the depot serving the next round trip. The question whether this is possible without shunting is equivalent to the question whether the type sequence of arriving trams can be sorted with one stack in such a way that we achieve as output the type sequence of departures.

The situation for more than one stack is illustrated in Figure 2.3.12. Shunting of trams may only be necessary for trams and departures assigned to the same stack. The resulting tram dispatch problem for several stacks is again a stack sorting problem but for a fixed number R of stacks, $R \geq 1$.

In this stack sorting problem, we assume that the sequence that has to be sorted is first completely assigned to the stack(s) before secondly the output sequence is satisfied.

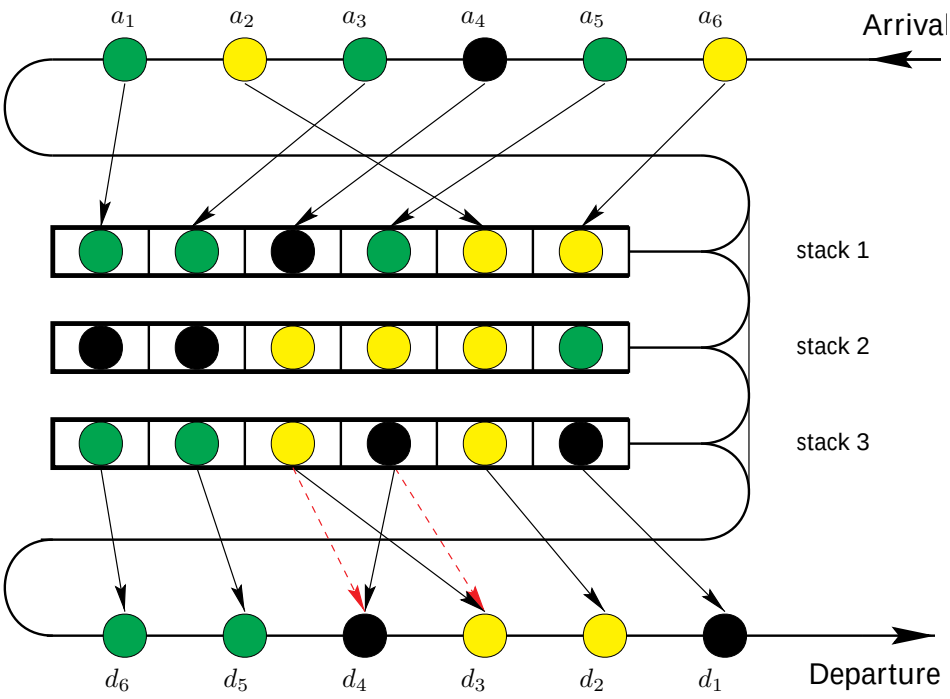

Figure 2.3.12: A model for the tram dispatch problem: Combination of arrivals and departures for three stacks.

2.4 Online and Real-Time Dispatch

In the daily dispatch process, the decision at which position a tram should be stored has to be found within a short time interval. Only a few minutes before a tram arrives at the depot, the dispatcher gets knowledge about the concrete arrival time of the tram and the implied arrival order. The transmission of this information differs for the transport companies. Usually, the dispatcher is informed via radio data transmission just before the tram arrives.

Due to the lack of space for arriving trams, the trams can only wait for a moment at the depot entrance before they enter the storage yard on the way to their standing position. Therefore, the decision to which position an arriving tram should be assigned has to be made within a time period of two to five minutes depending on the time gap between two arrivals. The information about the standing position should be transmitted to the driver immediately (or a few seconds) after arrival. These close time bounds require a decision in real-time based on incomplete knowledge about the arrival order.

2.4.1 Real-Time Decision Support Systems

Real-time decision support systems [SPG+97] have found application in several computer maintained decision support systems in local transport. Real-time decision support systems should provide a "quick" reaction on external events that change the situation which has been the basis for previous computations. Usually, the real-time decisions must be made under several limitations such as resource limitations, hardware specifications, time availability, and incomplete information. The fundamental components of such a decision system (according to [SPG+97]) are given by:

1. Information management: current update of incoming information
2. Situation assessment: evaluation of the situation, decision whether or not a reaction of the system is required (or should be proposed)
3. Evaluation of alternatives: checking the possible actions on the real-time event
4. Decision: determining an action (which includes the possibility of not reacting at all)

Real-time decision systems often result in semi-autonomous systems that support and assist human operators. In a few cases, they are efficient and secure enough to replace the human operator completely. In contrast to this, they usually require highly qualified personnel.

The decisions made by the decision support system may be of local effect, for instance for short-term actions. In this case, the actual solution is slightly modified and adapted to the real-time event. According to [SPG+97], we call such a decision **reactive planning**. Another possibility is an **incremental planning** which modifies the current solution more globally which results in a solution update. The third possibility is to make a complete revision of the solution found so far. This should be done when the observed situation differs significantly from the predicted state so that the current solution becomes ineffective. In [SPG+97], such a reaction is called **deliberative planning**.

The choice of how to react on real-time events depends on the time available for computation and on the effect of the real-time event.

In the considered tram dispatch problem in local transport, a reactive planning is advisable if we are able to react on the delay of a tram (and the implied change in the arrival sequence) by delaying or temporarily storing the corresponding trams whose dispatch is influenced by the delay.

Incremental planning should be preferred whenever the arrival sequence changes more significantly so that the assignment of trams to positions has to be changed for some trams and some stacks. In this case, the current solution has to be changed for the part influenced by the real-time effect.

A complete revision is advisable if there is sufficient time to compute such a revised solution and if the real-time effect causes a significant increase of the amount of shunting on arrival and at departure.

In Chapter 8, we will go into more details concerning real-time algorithms for the tram dispatch problem.

2.4.2 Online Tram Dispatch

A more theoretical approach to examine the aspect of incomplete and changing information in the tram dispatch problem is to consider the online version of the problem. In an online problem [BEY98, FW98], the information needed for the decision is revealed piecewise, for instance tram by tram.

A theoretical concept for investigations into the performance of online algorithms, called competitive analysis, has been introduced by Sleator and Tarjan [ST85]. In competitive analysis, we compare the solution value obtained by an online algorithm to the optimal value that an optimal algorithm would yield if it is provided with complete information. In Chapter 6, we go into more details concerning competitive analysis. In Chapter 7, we consider different online tram dispatching problems arising in storage yards.

In an online setting for tram dispatch, we first assume that we do not have any knowledge about the arrival sequence of trams. The trams arrive tram by tram. The next arrival becomes known just after the actual tram has been assigned to a depot position. In contrast to the real-time setting introduced in the last section, an online algorithm is not restricted in its computation time needed for its decision where to assign the actual tram. In another (more theoretical) online situation for the dispatch of trams to departures, we do not know what kind of tram of which type has to leave the depot as the next.

Consequently, we define the following settings for the tram dispatching problems:

1. **Real-Time Setting**: In the real-time problem, we know the planned arrival sequence. Based on a predetermined solution according to the actual schedule, we have to react on changes in the arrival sequence within a fixed short time interval.

2. **Online Setting (Arrival-Departure-Problem)**: In the online problem, the arrival sequence is given tram by tram. The departure sequence is known. The next arriving tram (or the type of the next arriving tram) is revealed just after the actual tram has been assigned to a depot position (and to a departure). The time needed to compute each decision is not limited. Additionally, every particular assignment of trams determined so far must not be changed in the remaining dispatch process.

3. **Online Setting (Departure-Problem)**: We assume that we are given a depot filled up with trams of given types. The departure sequence of trams is generated departure by departure. Before the next departure is revealed, we have to choose a tram stored in the depot that should serve the round trip corresponding to the actual departure. This tram is assumed to leave the depot directly. As in the above online setting, the time needed to compute each decision is not limited.

2.5 Dispatching in Terminus Stations

A related dispatch problem occurs at terminus railway stations where the train's locomotive has to be exchanged. Examples of such terminus stations are the central railway stations in Frankfurt and Munich, Germany. After the train has entered the station and stopped on the corresponding dead-end track, the locomotive is decoupled and another locomotive is coupled to the last coach. Then, the train can leave the station in the same direction from where it has arrived.

The locomotives that are available at terminus stations are stored in a holding yard from where they are moved to the terminus station. Depending on the engine power and other operational attributes, the locomotives are divided into groups of locomotives that are allowed to pull certain train configurations. Usually, these groups are formed by the engine types.

The holding yard of the terminus station consists of the same kinds of sidings as introduced above, mostly of dead-end sidings. Due to the limited space, the locomotives are stored one behind the other so that we are faced with a situation similar to the situation in the tram dispatching problem at storage yards in local transport.

The only difference between these two problems is that at terminus railway stations the arrival and departure of locomotives cannot be strictly divided into two parts, i.e., an arrival and departure part, as we have observed for storage yards in local transport. This is caused by the overlapping arrivals and departures of trains at terminus stations.

Hence, the resulting dispatching problem turns out to be a stack sorting problem where we cannot wait with satisfying the departure sequence until the last arrival has taken place. Instead of this, locomotives are allowed to leave the holding yard before the last arrival of the dispatch period. However, the locomotives have to wait until regular maintenance is done.

In Section 9.4, we will briefly discuss the dispatch problem for terminus railway stations in connection with a problem arising in container shipment. We will point out similarities between these two problems.

2.6 Dispatching in Container Logistics

In maritime container terminals, several dispatch problems have to be solved in order to serve the container ships in time. For each container ship, the planning process starts two days before it enters the port. At that time, the ship's current loading information is transmitted from the previously visited container terminal (port) to the considered terminal. Most of the new containers to be loaded onto the ship will arrive at the terminal within this two day period. For a large number of containers which have to be loaded onto the ship, the exact arrival time is not known.

Based on this incomplete information about the arrival data and subject to requirements provided by the ship owner, the planner has to prepare a preliminary stowage plan for the container ship. This stowage plan specifies which container has to be loaded into which bay position. In this plan, each container is identified by its container type, its discharge port, and its weight.

The shipping company provides the dispatcher at the terminal with a first stowage plan in which for each bay position the discharge port and the container weight are specified. In Section 9.4, the problem of how to determine such a first stowage plan is introduced. The result of the stowage planning process is a complete stowage plan in which to each bay position a container (satisfying the type, weight, and port requirements) is assigned.

The container vessel consists of a number of bays which can be partitioned into several stacks of up to ten positions. Based on the sequence in which the containers for each bay should be loaded, we obtain a sequence of "types" of containers that have to be loaded by and transported to the quay cranes in this specified order.

Given the complete stowage plan, the containers are loaded onto the ship in the following way. Each container is transported from its position at the container terminal to a quay crane that moves it to the specified bay position. In the terminal, the containers are stored in stacks of up to three containers. The transport is carried out by straddle carriers. In order to derive a good stowage plan, the planner should take into account the loading and transportation process.

Once again, the container loading and transport problem is a real-time problem. A high percentage of the containers to be loaded onto the ships arrive when the loading process has already begun. Consequently, the dispatcher has to update the transport and loading plan if containers are delivered too late.

For the container dispatch problem, we present a new integrated planning approach combining stowage and transportation planning (cf. Chapter 10). We model the just-in-time delivery of containers to the cranes by a mixed integer program and discuss some computational results for real-world data from one of the container terminals operated by HHLA, Hamburg, Germany. Moreover, we present computational results for a best fit heuristic method.

Chapter 3

Computational Complexity

In this chapter, we concentrate on theoretical aspects of the problem of dispatching trams in storage yards. Before we start to develop "efficient" algorithms in order to solve this problem, we discuss its computational complexity, i.e., the difficulty to solve a (combinatorial optimization) problem. The "efficiency" of a problem is usually measured by the worst-case computational time needed to solve it.

A general framework in which such a quantitative discussion can be done is the theory of computational complexity initiated by the works of Cook [Coo71] and Karp [Kar72]. Based on the concept of Turing machines, the theory of computational complexity enables us to distinguish between problems which are "easily" solvable and problems which are hard to solve. Although this concept is only a theoretical model, it has a great impact on the design and the analysis of algorithms developed to solve combinatorial optimization problems.

We start this chapter with a brief overview on the basic concepts of computational complexity. In Section 3.2 and Section 3.3, we give a more formal definition of the tram dispatching problem with or without shunting, prove $\mathcal{NP}$-hardness for both variations, and consider some special cases. Section 3.4 is concerned with a subproblem of the tram dispatching problem. As a subproblem, we consider the departure part of the dispatching process. We give a mathematical formulation for this departure dispatch problem and show that it is $\mathcal{NP}$-hard.

3.1 Preliminaries

In this section, we introduce the basic notions of computational complexity which we need for the discussions of the following sections. For our purposes it suffices to introduce some concepts of computational complexity in a relatively informal manner. For a more comprehensive presentation of these and further concepts we refer to the books of Garey and Johnson [GJ79] and Papadimitriou [Pap94].

Problems and Problem Instances

In the following, we distinguish between "problems" and "problem instances". A **problem** or **problem class** is a general question which is defined by a general description of all its parameters and a statement of which properties the **solution of the problem** has to satisfy. A **problem instance**, or more briefly an **instance**, is obtained if we specify all open parameters of the problem with particular values.

Definition 3.1.1: A **combinatorial optimization problem (class)** Π is defined by

- a general specification of the parameters of a problem instance,
- a statement of which properties a solution has to satisfy — usually expressed by a (finite) set Ω to which a solution x belongs if it satisfies all the properties, and
- an objective function $obj : \Omega \to \mathbb{Z}$ to be minimized (or maximized) over the set Ω.

The objective function obj measures the value of each solution. The goal is to determine an optimal solution with respect to the objective sense, i.e., a minimum or maximum solution. ❑

In the following, we restrict ourselves to minimization problems with integer valued objective functions.

As an example for a problem class, we may consider the classical **assignment problem** defined on a weighted complete bipartite graph $G = (V, V, V \times V)$ where $V = \{v_1, v_2, v_3, \ldots, v_n\}$. Each edge $(v_i, v_j) \in V \times V$ has a nonnegative edge weight c_{ij}. The question is to determine an **assignment** in this graph, i.e., a perfect matching or a minimal subset of edges that connects all the vertices of the first bipartition to all the vertices of the second bipartition. This matching should be chosen in such a way that the sum of the weights of the matching edges is minimized. In this problem, the number n and the edge weights are the open parameters that are to be specified in a problem instance. The statement to be satisfied is the matching property. The objective is to minimize the sum of the weights of the matching edges.

An example for a problem instance of this assignment problem may be given by $n = 3$ and the cost matrix

$$C = (c_{ij}) = \begin{pmatrix} 1 & 2 & 1 \\ 3 & 2 & 4 \\ 3 & 2 & 5 \end{pmatrix}.$$

The optimal solution of this problem is $\{(v_1, v_3), (v_2, v_2), (v_3, v_1)\}$ with weight 6. This solution can also be presented by an **assignment** or **permutation matrix**,

i.e., by an $(n \times n)$-dimensional 0-1-matrix with exactly one 1-element in each row and column. The solution of our example is described by the matrix

$$X = (x_{ij}) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

The assignment problem with linear sum objective function belongs to the class of "easily" solvable problems. For instance, it can be solved efficiently by the Hungarian algorithm [Kuh55] or by Tomizawa's algorithm [Tom71]. A variation of the classical assignment problem with linear sum objective function is the so-called **quadratic assignment problem**. In this problem, we search an assignment represented by its assignment matrix $X = (x_{ij})$ that minimizes the quadratic objective function

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ik}b_{jl}x_{ij}x_{kl}.$$

where $A = (a_{ik})$ and $B = (b_{jl})$ are $(n \times n)$-dimensional cost matrices. The quadratic assignment problem is to one of the hardest combinatorial optimization problems. Up to now, even with very sophisticated methods it is impossible to solve arbitrary problem instances of size $n \geq 30$ to optimality in reasonable time with state-of-the-art computers. We will return to quadratic assignment problems in Chapter 4. A comprehensive survey on various kinds of assignment problems can be found in Burkard et al.[Bur98, BÇ98].

The above example of the linear sum assignment problem and the quadratic assignment problem illustrates the border between easy solvable problems and problems which are hard to solve. Next, we return to the theoretical concepts of computational complexity and to methods which enable us to distinguish between "easy" and "hard" problems.

Decision Problems and Algorithms

The problems considered so far are combinatorial optimization problems. The solution consists of an assignment and an objective value. Other combinatorial problems, for instance the problem whether or not there is a perfect matching in an arbitrary bipartite graph, require only the answer "yes" or "no". Such problems are called **decision problems** and their solutions are called **answers**. Decision problems form the basis of computational complexity. In the next steps, we give the basic terminology of computational complexity for decision problems. Later in this section, we show how optimization problems can be handled within this concept.

For our purposes, it is sufficient to view an **algorithm** as a procedure which proceeds step-by-step to solve a problem. An algorithm is said to solve a problem if it produces a correct solution (an answer) for each instance of the problem class.

The performance of an algorithm is measured with respect to the "size" of the problem instances to be solved. In order to solve the problem for a given instance by a computer, we have to encode the instance. A possible encoding scheme is to encode the instance as a sequence of symbols (bits) over a fixed **alphabet** ($\{0,1\}$). For each problem instance I of a problem class Π, we define this size as the **encoding length**, i.e., the number of bits required to represent the actual input parameters of I in the (binary) encoding scheme. Recall that we restrict ourselves to problem data consisting of integral numbers. For decision problems the question whether or not an instance belongs to the class Π now turns out to be a question whether or not the encoded sequence of the given instance belongs to a certain class of — with respect to Π — correct sequences called **language**. Such a **recognition problem** can be solved by **Turing machines**[AHU75].

The **running time** of an algorithm for problem Π is the number of elementary operations which have to be executed to solve instance I. The running time of an algorithm ALG is usually called the **time complexity** of ALG and can be formally defined as a function $time_{\mathrm{ALG}} : \mathbb{N} \rightarrow \mathbb{N}$ which maps each instance size n to the maximum number of elementary operations that algorithm ALG needs for solving an instance of size n. We assume that each arithmetic operation, each comparison, and all further elementary steps can be executed in constant time. This model is called the **unit cost RAM model**. In fact, we are not interested in the exact value of $time_{\mathrm{ALG}}$ but rather in the rate of growth of $time_{\mathrm{ALG}}$ for increasing n. Therefore, we look for lower and upper bounds for this rate of growth and denote these bounds according to the O-Notation [AHU75, PS82].

We call an algorithm ALG **polynomial** if its time complexity $time_{\mathrm{ALG}}$ is bounded from above by a polynomial function in n. The class of problems which can be solved by a polynomial algorithm is denoted by $\mathcal{P}$.

The Classes $\mathcal{NP}$ and $\mathcal{NPC}$

For many interesting combinatorial optimization problems no polynomial algorithm is known so far. It is assumed that no such algorithm exists. However, there is a theoretical concept to handle a large class of those problems (see [PS82]).

Definition 3.1.2: Given a decision problem Π and a fixed finite alphabet with which the instances of Π are coded. Let † be a distinguished symbol of the alphabet marking the end of the encoded instance. The encoding length of I is denoted by $l(I)$.

We say that Π belongs to the class $\mathcal{NP}$ if there is an algorithm ALG and a polynomial p for which the following holds:

A given instance I of Π has the answer "yes" if and only if there is a sequence $S(I)$ of symbols (of the alphabet) with length less than $p(l(I))$. If I † $S(I)$ is submitted to ALG, then ALG must yield the answer "yes" after at most $p(l(I))$ steps. ❑

Obviously, the class $\mathcal{NP}$ includes the class $\mathcal{P}$. It is still an open question whether or not $\mathcal{NP} = \mathcal{P}$.

Definition 3.1.3: A problem Π is called **polynomially reducible** to a problem Π' if there is a polynomial algorithm that transforms Π into Π', i.e., that maps any instance of Π to an instance of Π' with the same answer. The corresponding polynomial algorithm is called a **polynomial reduction**. ❑

Polynomial reductions enable us to classify the problems in $\mathcal{NP}$. Note that polynomial reductions are reflexive and transitive.

Definition 3.1.4: A problem Π is said to be **$\mathcal{NP}$-hard** if every problem in $\mathcal{NP}$ is polynomially reducible to Π. If Π itself belongs to $\mathcal{NP}$, Π is called **$\mathcal{NP}$-complete**. The class of $\mathcal{NP}$-complete problems is denoted by $\mathcal{NPC}$. ❑

Any two problems in $\mathcal{NPC}$ are polynomially reducible to each other. Thus, the class $\mathcal{NPC}$ forms a equivalence class with respect to polynomial reducibility [GJ79]. Cook [Coo71] showed that a particular problem in logic, called the **satisfiability problem**, is $\mathcal{NP}$-complete. Based on this problem, a large number of decision problems have been shown to be $\mathcal{NP}$-complete. A problem Π is shown to be $\mathcal{NP}$-complete for instance if we can polynomially reduce a problem known to be $\mathcal{NP}$-complete to Π. An ongoing guide containing an updated list of $\mathcal{NP}$-complete problems can be found in [Joh90, Joh92].

The problems in $\mathcal{NPC}$ are the "hardest" problems in $\mathcal{NP}$. Since each problem of $\mathcal{NP}$ can be reduced to every problem in $\mathcal{NPC}$, we can solve every problem of $\mathcal{NP}$ by an algorithm that solves a particular problem of $\mathcal{NPC}$ and by the corresponding algorithms that carry out the polynomial reductions. Hence, if a problem in $\mathcal{NPC}$ is solvable in polynomial time, then all problems of $\mathcal{NPC}$ are solvable in polynomial time, too.

Optimization Problems and $\mathcal{NP}$-completeness

Strictly speaking, $\mathcal{NPC}$ consists only of decision problems and contains no (combinatorial) optimization problem. An optimization problem can be transformed into a decision problem in the following way. We concentrate on minimization problems. For maximization problems, the following concept can be adapted in an analogous way.

We define the corresponding decision problem by asking if there is a solution for this instance with an objective value less than or equal to a constant K. Assuming that the objective value can be evaluated in polynomial time, the decision problem is no harder than the original optimization. Hence, any negative result proved about the decision problem applies to the optimization problem as well. We call an optimization problem $\mathcal{NP}$-hard, if the corresponding decision problem is $\mathcal{NP}$-hard (or $\mathcal{NP}$-complete).

3.2 The Tram Dispatching Problem

Before we start our complexity investigations, we give a more formal definition of the tram dispatching problem.

Definition 3.2.5: The **Tram Dispatching Problem (TDP)** is given by the following instance and the following objective.

Instance:
- a set $\mathcal{A} = \{a_1, \ldots, a_N\}$ of arriving trams
- a set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of departures
- a set $\mathcal{P}$ of N depot positions located in R stacks of sizes P_r, $1 \leq r \leq R$, where the P_r positions in stack $\mathcal{P}_r \subseteq \mathcal{P}$ are numbered consecutively from the bottom to the top and denoted by $\sum_{i=1}^{r-1} P_i + 1, \ldots, \sum_{i=1}^{r} P_i$,
- a mapping $t : \mathcal{A} \cup \mathcal{D} \rightarrow \{\tau_1, \ldots, \tau_T\}$ which assigns to each element of $\mathcal{A}$ and $\mathcal{D}$ a type in $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$.

Objective: Find two assignments $\pi_X : \mathcal{A} \rightarrow \mathcal{P}$ and $\pi_Y : \mathcal{D} \rightarrow \mathcal{P}$ represented by the two assignment matrices $X = (x_{iq})$ and $Y = (y_{jq})$ satisfying the property:

$(*)$ $\forall\, p_q \in \mathcal{P} :\ (x_{iq} = 1 \ \wedge y_{jq} = 1) \quad \Rightarrow \quad t(a_i) = t(d_j)$,

and minimizing the cardinality of the set SM where

$$\begin{aligned} SM \quad = \quad & \bigcup_{r=1}^{R} \{(a_i, a_k) \in \mathcal{A} \times \mathcal{A} \mid \ x_{iq} = 1, x_{kl} = 1, i < k, \\ & \qquad\qquad p_q, p_l \in \mathcal{P}_r, \text{and } q > l\} \\ \cup \quad & \bigcup_{r=1}^{R} \{(d_j, d_k) \in \mathcal{D} \times \mathcal{D} \mid \ y_{jq} = 1, y_{kl} = 1, j < k, \\ & \qquad\qquad p_q, p_l \in \mathcal{P}_r, \text{and } q < l\}. \end{aligned}$$

The **decision problem corresponding to TDP** is the problem of determining for a given instance whether or not feasible assignments π_X and π_Y exist such that the cardinality of SM is bounded from above by K.

A special case of such a decision problem is the problem where we search for two feasible assignments π_X and π_Y such that SM is empty. We refer to this particular decision problem as **zero shunting TDP (0-TDP)**. ❑

The set SM can be considered as the set of pairs of trams which have to be shunted in order to assign the trams to the stacks or to the departures as given by the assignments π_X and π_Y.

The assignments $\pi_X : \mathcal{A} \rightarrow \mathcal{P}$ and $\pi_Y : \mathcal{D} \rightarrow \mathcal{P}$ are said to be **feasible for TDP** if they satisfy property $(*)$. Since $\mathcal{A}$ and $\mathcal{D}$ are of the same cardinality, $(*)$ requires that

$$\forall \tau \in \mathcal{T} : \quad \mid \{a \in \mathcal{A} \mid t(a) = \tau\} \mid = \mid \{d \in \mathcal{D} \mid t(d) = \tau\} \mid$$

3.2.1 Complexity of TDP

In the following, we prove that the tram dispatching problem is $\mathcal{NP}$-hard by giving a polynomial reduction from the three dimensional matching problem to

0-TDP. The three dimensional matching problem, one of the well-known $\mathcal{NP}$-complete problems, is defined as follows [GJ79]:

Definition 3.2.6: The decision problem

Instance: A set $S \subseteq \mathsf{X} \times \mathsf{Y} \times \mathsf{Z}$, where X, Y, and Z are finite sets of cardinality a and pairwise disjoint.

Question: Does S contain a subset S' so that $\mid S' \mid = a$ and no two elements of S' agree in any coordinate?

is called **Three Dimensional Matching Problem (3DM)**. ❑

Theorem 3.2.7: 3DM is $\mathcal{NP}$-complete.

Proof: See Garey and Johnson [GJ79]. ❑

In the TDP, we want to assign each element of $\mathcal{A}$ to an element of $\mathcal{P}$ and each element of $\mathcal{D}$ to an element of $\mathcal{P}$. The connection between the two assignments π_X and π_Y can be represented by triples $(a_i, p_q, d_j) \in \mathcal{A} \times \mathcal{P} \times \mathcal{D}$. Each pair of assignments π_X and π_Y defines a subset S' of $\mathcal{A} \times \mathcal{P} \times \mathcal{D}$. This subset S' is defined by

$$(a_i, p_q, d_j) \in S' :\Leftrightarrow x_{iq} = 1 \text{ and } y_{jq} = 1 \text{ for } a_i \in \mathcal{A}, p_q \in \mathcal{P}, \text{ and } d_j \in \mathcal{D}.$$

where $X = (x_{iq})$ and $Y = (y_{jq})$ are the corresponding assignment matrices.

Theorem 3.2.8: 0-TDP is $\mathcal{NP}$-complete.

Proof: We consider the following decision problem: Given an instance of TDP, is there an assignment of arriving trams to positions and to departures which does not require any shunting?

For a given assignment, we can answer this question by checking if shunting is necessary for the trams stored in each stack. Since this can be done in polynomial time, 0-TDP is in $\mathcal{NP}$.

We prove $\mathcal{NP}$-completeness of 0-TDP by giving a polynomial reduction from 3DM to 0-TDP.

Given an instance of 3DM, we construct an instance of TDP as follows.

Let b denote the cardinality of S and let $\mathsf{X} = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_a\}$, $\mathsf{Y} = \{\mathsf{y}_1, \mathsf{y}_2, \ldots, \mathsf{y}_a\}$, and $\mathsf{Z} = \{\mathsf{z}_1, \mathsf{z}_2, \ldots, \mathsf{z}_a\}$. Without loss of generality, $b > a$. (If $b < a$, no three-dimensional matching exists. If $b = a$, 3DM is trivially decidable in polynomial time.)

The corresponding instance of TDP is defined as follows:

> We consider $4b$ arrivals and $4b$ departures and a depot of b stacks, each of length 4. For each element of X, Y, and Z, we introduce a type that we identify by the corresponding element of X, Y, and Z.

We number the b tuples in S consecutively from 1 to b. For each tuple $s_i = (\mathsf{x}, \mathsf{y}, \mathsf{z}) \in S$, we define an unique type D_i so that the TDP instance consists of $3a + b$ types of trams.

Each tuple of S is identified by $s_i = (\mathsf{x}_{j_i}, \mathsf{y}_{k_i}, \mathsf{z}_{l_i})$, where i denotes the position of s_i with respect to the above numbering. For each tuple s_i, we have 4 trams of type D_i, x_{j_i}, y_{k_i}, and z_{l_i}.

We define the arrival sequence $\mathcal{A} = \{a_1, a_2, \ldots, a_{4b}\}$ as follows:

$$t(a_i) = \begin{cases} \mathsf{z}_{l_v} & \text{if } i = 4(v-1)+1 = 4v-3 \\ \mathsf{y}_{k_v} & \text{if } i = 4(v-1)+2 = 4v-2 \\ \mathsf{x}_{j_v} & \text{if } i = 4(v-1)+3 = 4v-1 \\ D_v & \text{if } i = 4v \end{cases}, \quad 1 \le v \le b.$$

The departure sequence $\mathcal{D} = \{d_1, d_2, \ldots, d_{4b}\}$ is defined by:

$$t(d_i) = \begin{cases} D_i & \text{if } 1 \le i \le b \\ \mathsf{x}_j & \text{if } i = b + j, 1 \le j \le a \\ \mathsf{y}_j & \text{if } i = a + b + j, 1 \le j \le a \\ \mathsf{z}_j & \text{if } i = 2a + b + j, 1 \le j \le a \\ \mathsf{x}_{F_\mathsf{X}(i)} & \text{if } 3a + b + 1 \le i \le 2a + 2b \\ \mathsf{y}_{F_\mathsf{Y}(i)} & \text{if } 2a + 2b + 1 \le i \le a + 3b \\ \mathsf{z}_{F_\mathsf{Z}(i)} & \text{if } a + 3b + 1 \le i \le 4b \end{cases}$$

The mapping $F_\mathsf{X} : \{3a + b + 1, \ldots, 2a + 2b\} \to \{1, \ldots, a\}$ is (arbitrarily) chosen in such a way that

$$| \{i \mid \mathsf{x}_{F_\mathsf{X}(i)} = \mathsf{x}_j\} | \quad = \quad | \{s_i \in S \mid \mathsf{x}_{j_i} = \mathsf{x}_j\} | - 1 \quad \text{for all } \mathsf{x}_j \in \mathsf{X}$$

The mappings F_Y and F_Z are defined analogously for $\mathsf{y}_k \in \mathsf{Y}$ and $\mathsf{z}_l \in \mathsf{Z}$.

Consequently, the trams arrive in the following type sequence starting with a_1:

$$(\mathsf{z}_{l_1}, \mathsf{y}_{k_1}, \mathsf{x}_{j_1}, D_1, \mathsf{z}_{l_2}, \mathsf{y}_{k_2}, \mathsf{x}_{j_2}, D_2, \ldots, \mathsf{z}_{l_b}, \mathsf{y}_{k_b}, \mathsf{x}_{j_b}, D_b).$$

Beginning with d_1, the first $3a + b$ departures are given by

$$(D_1, D_2, \ldots, D_b, \mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_a, \mathsf{y}_1, \mathsf{y}_2, \ldots, \mathsf{y}_a, \mathsf{z}_1, \mathsf{z}_2, \ldots, \mathsf{z}_a).$$

The sequence of the remaining $3(b-a)$ departures starts with $b-a$ departures of type $\mathsf{x} \in \mathsf{X}$ followed by $b-a$ departures of type $\mathsf{y} \in \mathsf{Y}$ and is completed with $b-a$ departures of type $\mathsf{z} \in \mathsf{Z}$. The types are chosen by F_X, F_Y, and F_Z in such a way that there is a one to one correspondence between arriving trams of departures of same type.

We refer to the first b departures $d_1, d_2, \ldots, d_b$ as the "*leading departures*". The departures $d_{b+1}, \ldots, d_{b+3a}$ are called "*matching departures*" and the remaining $3(b-a)$ departures "*non-matching departures*".

Since all trams of type $D_1, \ldots, D_b$ arrive in the same order as they depart, shunting cannot be avoided if two trams of these types are assigned to the same stack. In a shunting-free solution of TDP, to each stack exactly one tram of type D_i is assigned. Otherwise, at least two of these trams must be shunted. All these trams have to leave the depot as first. In each stack, the tram of type D_i must be assigned to the top position to avoid shunting. This implies that, in a shunting-free and type preserving solution of TDP, below the tram of a certain type D_k the corresponding trams of type x_{j_i}, y_{k_i}, and z_{l_i} of the same tuple s_i have to be assigned. (Note that every fourth arrival is of type D_i).

Proposition 3.2.9: The instance of 3DM contains a three dimensional matching if and only if the constructed instance of TDP has a solution without shunting.

Proof: "⇒": If the 3DM instance admits a three-dimensional matching S', the following solution of the TDP does not require shunting.

The trams corresponding to the same tuple s_i are assigned to the same stack in the order of their arrival. We number the positions in each stack consecutively from the bottom to the top. We assign the tram of type z_{l_i} to the bottom position, the tram of type y_{k_i} to the second position, the tram of type x_{j_i} to the third position, and the tram of type D_i to the top position. All these assignments are possible without shunting.

In order to serve the leading departures, all trams of type D_i, $1 \leq i \leq b$, leave the stacks without shunting. Next, we have to serve the $3a$ matching departures. For each $1 \leq i \leq a$, exactly one tram of each type x_i must leave the depot, followed by exactly one tram of each type y_i and exactly one tram of each type z_i.

Since the instance admits a three dimensional matching, there are a stacks in which all trams corresponding to elements of S' are stored. These trams are needed to serve the matching departures. All these trams are able to leave the depot without shunting.

Finally, we remove the remaining $3(b-a)$ trams from the stacks beginning with the trams of type x_i, followed by the trams of type y_i and concluding with the trams of type z_i. Once again, no shunting movement is necessary to serve these departures.

Hence, if the 3DM instance contains a three dimensional matching, then for the corresponding TDP-instance a shunting-free and type preserving solution exists.

"⇐": For the instance of TDP, we assume that now we know a shunting-free and type preserving solution. As we have stated above, in such a solution all trams

of types D_i, $1 \leq i \leq b$, must be assigned to different stacks. Moreover, in each stack a tram of type D_i has to be assigned to the top position. Since every fourth arrival is of D-type, there is no other possible shunting-free assignment of trams to positions than a solution of that kind already introduced in the first part of the proof.

The departures start with the leading departures of type D_1 to D_b. Then, the only possibility to serve the next $3a$ departures without shunting is to empty a stacks. Since exactly one tram of type x_i, one tram of type y_i, and one tram of type z_i for $1 \leq i \leq a$ is taken to serve the corresponding departures, the elements of these a stacks correspond to the tuples that form a three dimensional matching S' in the 3DM instance. ❑

The above construction implies a polynomial reduction from 3DM to 0-TDP. Consequently, 0-TDP is $\mathcal{NP}$-complete. ❑

Corollary 3.2.10: TDP is $\mathcal{NP}$-hard.

Proof: 0-TDP is a special case of the general decision problem corresponding to TDP. Hence, this decision problem is also $\mathcal{NP}$-complete and TDP is $\mathcal{NP}$-hard. ❑

In the proof of Theorem 3.2.8, we have constructed a reduction from an arbitrary instance of 3DM to an instance of TDP consisting of stacks of size four.

Corollary 3.2.11: 0-TDP remains $\mathcal{NP}$-complete even if we restrict ourselves to instances of fixed stack size equal to L, $L \geq 4$.

Proof: The case $L = 4$ follows immediately from the proof of Theorem 3.2.8. If $L > 4$, we define for each tuple $s_i \in S$ exactly $L - 3$ trams of type D_i. The departure sequence $\tilde{\mathcal{D}}$ is adapted from $\mathcal{D}$ in such a way that $\tilde{t}(d_i) = D_k$ for $k = \lfloor \frac{i}{L-3} \rfloor$, $1 \leq i \leq (L-3)b$, and $\tilde{t}(d_{(L-3)b+i}) = t(d_{b+i})$. ❑

The following corollary is analogous to Corollary 3.2.10.

Corollary 3.2.12: TDP is $\mathcal{NP}$-hard even if we restrict ourselves to instances of fixed stack size equal to L, $L \geq 4$.

Next, we investigate some of the remaining cases.

Theorem 3.2.13: TDP is solvable in polynomial time if we restrict ourselves to instances with stack size 2 and N trams of pairwise different types.

Proof: We prove the theorem by giving an algorithm which solves such restricted instances of TDP in polynomial time.

To compute the minimum number of shunting movements necessary for an instance of TDP with uniform stack size 2 and N trams of pairwise different types, we construct a weighted graph $G = (V, E)$ where G is a complete graph without loops (u, u) for all $u \in V$.

The set $\mathcal{A}$ of arrivals consists of N trams of pairwise different types. The same holds for the set $\mathcal{D}$ of departures. Each possible combination of two arrivals that are assigned to the same stack can be identified by the pair $(a_i, a_k) \in \mathcal{A} \times \mathcal{A}$. The pair (a_i, a_k) corresponds to a stack in which a_i is on top of a_k. Such an assignment requires a shunting movement if and only if $i < k$.

Since we restrict ourselves to instances of N types, we have a bijection from $\mathcal{A}$ to $\mathcal{D}$ given by the types of the elements of $\mathcal{A}$ and $\mathcal{D}$. Tram $a_i \in \mathcal{A}$ is assigned to the same position as departure $d_j \in \mathcal{D}$ if and only if $t(a_i) = t(d_j)$. Since there are N trams of different types, there is only a unique departure $d_j = d_{j(i)}$ to which the type of a_i fits and vice versa. Therefore, we define the node set V to be the set of the N pairs $(a_i, d_{j(i)})$ with $t(a_i) = t(d_{j(i)})$.

In the next step, we assume that we assign $a_i \in \mathcal{A}$ and $a_k \in \mathcal{A}$ to the same stack. We denote by $d_{j(i)}$ and $d_{j(k)}$ the corresponding departures. Then, the following holds:

- No shunting movement is required if and only if $i > k$ and $j < l$.
- Exactly one shunting movement is necessary if either $i < k$ or $j(i) > j(k)$.
- Two shunting movements are necessary if $i < k$ and $j(i) > j(k)$.

Note that the last case does not hold for an optimal assignment, since a better assignment can be achieved by simply switching the positions for a_i and a_k. The edge weight of (u, v) is defined as the number of shunting movements required if $u = (a_i, d_{j(i)})$ and $v = (a_k, d_{j(k)})$ are assigned to the same stack. Then, the minimum number of shunting movements required for this instance of the TDP is equal to the weight of the minimum perfect matching in G which can be determined in polynomial-time (cf. [AMO93]).

An edge between $(a_i, d_{j(i)})$ and $(a_k, d_{j(k)})$ in an optimal perfect matching in G corresponds to a stack to which a_i and a_j are assigned. a_i and a_j are assigned in such a way that either one shunting movement is required or the assignment is possible without shunting. For the departure, no shunting is required. In case that N is odd or in case that there are stacks of length 1, we introduce a dummy node for each stack of length 1. The weight on edges adjacent to a dummy node is zero. ❑

Remark 3.2.14: The computational complexity of TDP remains open for the following two cases.

- TDP with stack size 2 and less than N types and
- TDP with stack size 3.

3.3 Dispatching Trams without Shunting

Before we start our complexity investigations, we give a formal definition of the tram dispatching problem without shunting to which we refer as the **type mismatch problem**.

Definition 3.3.15: The **Type Mismatch Problem (TMP)** is given by the following instance and the following objective.

Instance:
- a set $\mathcal{A} = \{a_1, \ldots, a_N\}$ of arriving trams
- a set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of departures
- a set of N depot positions $\mathcal{P}$ located in R stacks of sizes P_r, $1 \leq r \leq R$, where the P_r positions in stack $\mathcal{P}_r \subset \mathcal{P}$ are numbered consecutively from the bottom to the top and denoted by $\sum_{i=1}^{r-1} P_i + 1, \ldots, \sum_{i=1}^{r} P_i$,
- a mapping $t : \mathcal{A} \cup \mathcal{D} \to \{\tau_1, \ldots, \tau_T\}$ which assigns to each element of $\mathcal{A}$ and $\mathcal{D}$ a type in $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$.

Objective: Find two assignments $\pi_X : \mathcal{A} \to \mathcal{P}$ and $\pi_Y : \mathcal{D} \to \mathcal{P}$ represented by the two assignment matrices $X = (x_{iq})$ and $Y = (y_{jq})$ for which

$(*)$ the set SM is the empty set where

$$\begin{aligned} SM \quad = \quad & \bigcup_{r=1}^{R} \{(a_i, a_k) \in \mathcal{A} \times \mathcal{A} \mid \quad x_{iq} = 1, x_{kl} = 1, i < k, \\ & \qquad\qquad\qquad\qquad\quad p_q, p_l \in \mathcal{P}_r, \text{and } q > l\} \\ \cup \quad & \bigcup_{r=1}^{R} \{(d_j, d_k) \in \mathcal{D} \times \mathcal{D} \mid \quad y_{jq} = 1, y_{kl} = 1, j < k, \\ & \qquad\qquad\qquad\qquad\quad p_q, p_l \in \mathcal{P}_r, \text{and } q < l\}. \end{aligned}$$

and the number of pairs $(a_i, d_j) \in \mathcal{A} \times \mathcal{D}$ with $\pi_X(a_i) = \pi_Y(d_j)$ and $t(a_i) \neq t(d_j)$ is minimized.

The assignments $\pi_X : \mathcal{A} \to \mathcal{P}$ and $\pi_Y : \mathcal{D} \to \mathcal{P}$ are said to be **feasible for TMP** if they satisfy property $(*)$, i.e., if they avoid shunting completely.

The **decision problem corresponding to the TMP** is the problem of determining for a given instance whether or not feasible assignments π_X and π_Y exist with at most K pairs $(a_i, d_j) \in \mathcal{A} \times \mathcal{D}$ with $\pi_X(a_i) = \pi_Y(d_j)$ and $t(a_i) \neq t(d_j)$. We refer to the particular decision problem with $K = 0$ as (0-TMP). ❑

Analogously to Definition 3.2.5, SM represents the set of pairs of trams which have to be shunted in order to assign the trams to the stacks or to the departures as given by the assignments π_X and π_Y. Since we seek for feasible assignments which avoid shunting, SM has to be empty.

By construction, 0-TMP and 0-TDP are equivalent problems. Hence, we can state the following theorem.

Theorem 3.3.16: 0-TMP is $\mathcal{NP}$-complete.

Proof: This result follows immediately from the proof of Theorem 3.2.8. ❑

Consequently, we conclude that

Corollary 3.3.17: TMP is $\mathcal{NP}$-hard.

Proof: Since 0-TMP is a special case of the decision problem corresponding to TDP, if follows that this decision problem is also $\mathcal{NP}$-complete. This implies that TMP is $\mathcal{NP}$-hard. ❑

3.4 Dispatching Trams to the Departures

In the following, we focus on a subproblem of TDP. In this subproblem, we assume that we are given an assignment π_X by which the arriving trams are already assigned to the stacks. The problem to be solved is to answer the question as to how many trams have to shunted in order to serve a given departure sequence.

Definition 3.4.18: The **Departure Dispatch Problem (DTDP)** is given by the following instance and the following question:

Instance:
- a set $\mathcal{A} = \{a_1, \ldots, a_N\}$ of arriving trams
- a set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of departures
- a set of N depot positions $\mathcal{P}$ located in R stacks of sizes P_r, $1 \leq r \leq R$, where the P_r positions in stack $\mathcal{P}_r \subset \mathcal{P}$ are numbered consecutively from the bottom to the top and denoted by $\sum_{i=1}^{r-1} P_i + 1, \ldots, \sum_{i=1}^{r} P_i$,
- a mapping $t : \mathcal{A} \cup \mathcal{D} \rightarrow \{\tau_1, \ldots, \tau_T\}$ which assigns to each element of $\mathcal{A}$ and $\mathcal{D}$ a type in $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$.
- An assignment $\pi_X : \mathcal{A} \rightarrow \mathcal{P} = \bigcup_{r=1}^{R} \mathcal{P}_r$ represented by the assignment matrix X which maps a set $\mathcal{A}$ of N trams to N positions

Question: Find an assignment $\pi_Y : \mathcal{D} \rightarrow \mathcal{P}$ represented by the assignment matrix Y satisfying

$(*)$ $\forall\, p_q \in \mathcal{P} :\ (x_{iq} = 1 \wedge y_{jq} = 1) \quad \Rightarrow \quad t(a_i) = t(d_j)$

and minimizing the cardinality of SMD where

$$SMD \quad = \quad \bigcup_{r=1}^{R} \{(d_j, d_k) \in \mathcal{D} \times \mathcal{D} \mid \ y_{jq} = 1, y_{kl} = 1, j < k, \ p_q, p_l \in \mathcal{P}_r, \text{and } q < l\}.$$

Analogous to Definition 3.2.5 and Definition 3.3.15, we define the corresponding decision problem to DTDP as the problem of finding a feasible assignment for which the cardinality of SMD is bounded from above by K. The decision problem for $K = 0$ is denoted by **0-DTDP**. ❑

The set SMD represents the set of pairs of trams that have to be shunted in order to leave the depot in the order given by the assignment π_Y.

Theorem 3.4.19: 0-DTDP is $\mathcal{NP}$-complete.

Proof: Given an assignment π_Y, it can be verified in polynomial time if π_Y satisfies $(*)$ and if SMD is empty. Thus, 0-DTDP is in $\mathcal{NP}$. We prove that 0-DTDP is $\mathcal{NP}$-complete by giving a polynomial reduction from 3DM to 0-DTDP.

By Definition 3.2.6, an instance of 3DM is given as follows.

Instance: Three sets X,Y, and Z, each of cardinality a, and a collection S of b tuples $s = (\mathsf{x}_s, \mathsf{y}_s, \mathsf{z}_s) \in \mathsf{X} \times \mathsf{Y} \times \mathsf{Z}$.

Question: Is there a subset $S' \subseteq S$ of a tuples $s_1, \ldots, s_a$, such that $\mathsf{X} = \{\mathsf{x}_{s_i} \mid i = 1, \ldots, a\}$, $\mathsf{Y} = \{\mathsf{y}_{s_i} \mid i = 1, \ldots, a\}$, and $\mathsf{Z} = \{\mathsf{z}_{s_i} \mid i = 1, \ldots, a\}$?

Given an instance of 3DM, we construct an instance of DTDP as follows. We denote by b the cardinality of S. Without loss of generality, we again assume that $b > a$.

We consider b stacks, each stack of size $P_r = 3$, $1 \leq r \leq b$. For each element of X, Y, and Z we define a type τ that we identify by the corresponding element of X, Y, and Z. Hence, $\mathcal{T}$ consists of $3a$ different types.

We number the tuples of S consecutively from 1 to b. For each tuple $s_i = (\mathsf{x}_{j_i}, \mathsf{y}_{k_i}, \mathsf{z}_{l_i}) \in S$, we introduce three trams $a_{3i}, a_{3i-1}, a_{3i-2}$ of type x_{j_i}, y_{k_i}, and z_{l_i}. We assign the trams to stack i in the following way.

Tram a_{3i-2} of type z_{j_i} is assigned to the bottom position of stack i.
Tram a_{3i-1} of type y_{k_i} is assigned to the middle position of stack i.
Tram a_{3i} of type x_{l_i} is assigned to the top position of stack i.

The stack positions are numbered consecutively from 1 to N beginning by stack 1 and ending with stack R. Hence, a_i is assigned to position p_i, $1 \leq i \leq N$.

The corresponding assignment matrix X is the $(N \times N)$-dimensional identity matrix and π_X is the assignment that maps p_i to d_i for all $1 \leq i \leq N$.

The types of the departures $\mathcal{D} = \{d_1, \ldots, d_{3b}\}$ are given by

$$t(d_i) = \begin{cases} \mathsf{x}_i & \text{for} & 1 \leq i \leq a \\ \mathsf{y}_{i-a} & \text{for} & a+1 \leq i \leq 2a \\ \mathsf{z}_{i-2a} & \text{for} & 2a+1 \leq i \leq 3a \\ \mathsf{x}_{F_{\mathsf{X}}(i)} & \text{for} & 3a+1 \leq i \leq 2a+b \\ \mathsf{y}_{F_{\mathsf{Y}}(i)} & \text{for} & 2a+b+1 \leq i \leq a+2b \\ \mathsf{z}_{F_{\mathsf{Z}}(i)} & \text{for} & a+2b+1 \leq i \leq 3b \end{cases},$$

where F_{X}, F_{Y}, and F_{Z} are defined in an analogous way as in the proof of Theorem 3.2.8. As in Theorem 3.2.8, the functions F_{X}, F_{Y}, and F_{Z} guarantee that there is a one-to-one correspondence between the trams and departures of the same type. Hence, property $(*)$ can be satisfied.

We prove the theorem by showing that the 3DM instance contains a three dimensional matching if and only if for the constructed DTDP-instance an assignment without shunting exists.
"⇒": Given a three dimensional matching, there are a stacks corresponding to the tuples of the three dimensional matching. In each of these stacks the x-type tram is at the top position followed by the y-type tram and the z-type tram at the bottom. Hence, the first $3a$ departures of $\mathcal{D}$ can be served by these trams without shunting.

The next $3(b-a)$ departures require trams in the following order. At first $b-a$ trams of x-type must be assigned, followed by $b-a$ trams of y-type and $b-a$ trams of z-type. In the stacks, all trams of x-type are on-top of the trams of y-type which itself are on-top of the trams of z-type. Hence, to all these departures, trams of suitable type can be assigned without shunting.

Consequently, if the 3DM instance contains a three dimensional matching, then for the constructed DTDP-instance a shunting-free assignment of trams to departures exists.
"⇐": Now, we assume that we know a shunting-free assignment for the DTDP-instance. To the first a departures, a trams of types $\mathsf{x}_1, \ldots, \mathsf{x}_a$ are assigned. The next a departures of types $\mathsf{y}_1, \ldots, \mathsf{y}_a$ can be served without shunting. Hence, all the trams assigned to these departures must be in the same a stacks from which the first a departures are served. Otherwise, we would need a shunting movement of a tram of y-type with a tram of x-type. We can apply the same argumentation to the next a departures of types $\mathsf{z}_1, \ldots, \mathsf{z}_a$.

After the first $3a$ departures are served, a stacks are empty. These a stacks contained exactly one tram of each type corresponding to an element in X, Y, and Z. This implies that the tuples corresponding to these stacks define a three dimensional matching for the 3DM instance.

Since the above construction implies a polynomial reduction from 3DM to 0-DTDP, 0-DTDP is $\mathcal{NP}$-complete. ❑

Since 0-DTDP is a special case of the decision problem corresponding to DTDP, we achieve the following corollary.

Corollary 3.4.20: DTDP is $\mathcal{NP}$-hard.

Next, we investigate the decision problem 0-DTDP restricted to a fixed number of tram types used. Obviously, 0-DTDP (and DTDP) is trivially solvable in linear time if $\mathcal{T}$ consists of only one type. For two or more types, this changes drastically. Even for instances with two types, 0-DTDP turns out to be $\mathcal{NP}$-complete.

Theorem 3.4.21: 0-DTDP remains $\mathcal{NP}$-complete even if we restrict ourselves to instances with a fixed number of types $T \geq 2$.

Proof: We prove this theorem by considering only the case $T = 2$. We give a polynomial reduction from 0-DTDP with an arbitrary number $\tilde{T}$ of tram types to 0-DTDP with $T = 2$ tram types.

We consider an instance $\tilde{I}$ of 0-DTDP with N trams, R stacks, and $\tilde{T}$ types. Furthermore, let $\tilde{\mathcal{D}} = \{\tilde{d}_1, \ldots, \tilde{d}_N\}$ be the set of departures and $\tilde{\mathcal{T}} = \{\tau_1, \ldots, \tau_{\tilde{T}}\}$ be the set of types. We represent each type by a certain binary string. We will show that we can replace each departure $\tilde{d} \in \tilde{\mathcal{D}}$ of type $\tau \in \tilde{\mathcal{T}}$ by a sequence of departures of types 0 and 1 corresponding to 0-1-strings.

In the following, we denote by [0] a sequence of $\lceil \log_2(\tilde{T}) \rceil$ zeros and by [1] a sequence of $\lceil \log_2(\tilde{T}) \rceil$ ones.

The 0-1-string for each type τ consists of three parts denoted by head, code, and tail. For each type, the head is given by the sequence

$$[0][0]001.$$

The code itself is the binary representation of the integer value i, i.e., of the index of type $\tau_i \in \mathcal{T}$ added to a sequence of leading zeroes such that all codes have the same length $\lceil \log_2(\tilde{T}) \rceil$. The tail of every type is given by the sequence

$$0[1]1.$$

Evidently, the length of the 0-1-string encoding type τ and the time needed to determine this string are polynomial in the input size. Using this encoding scheme, we encode the types of trams stored in the stacks as well as the types of the departure sequence $\tilde{\mathcal{D}}$.

We construct a set $\mathcal{A}$ of "trams" stored in R stacks $\mathcal{P}_1, \ldots, \mathcal{P}_R$ and a set $\mathcal{D}$ of departures. The encoding length of each tram type is $4\lceil \log_2(\tilde{T}) \rceil + 5$. The lengths of each stack $\mathcal{P}_r$ $(1 \leq r \leq R)$ as well as the lengths of $\mathcal{A}$ and $\mathcal{D}$ increase by the same factor compared to the corresponding lengths in the instance of 0-DTDP. Consequently, these new lengths are bounded from above by a function which is logarithmic in the number of types $\tilde{T}$ and linear in N.

The types of the first $4\lceil \log_2(\tilde{T}) \rceil + 5$ departures in $\mathcal{D}$ are 0 or 1 in the order corresponding to the string encoding of the type of first departure $\tilde{d}_1 \in \tilde{\mathcal{D}}$. The first departure of $\mathcal{D}$ is of type 0 corresponding to the first 0 in the head of the string encoding the type of the first departure in $\tilde{\mathcal{D}}$. The following departure types are defined in an analogous way according to the strings encoding the types of $d_2, \ldots, d_N$. The same holds for the encoding of each stack element. At the first (top-most) $4\lceil \log_2(\tilde{T}) \rceil + 5$ positions of each stack there are "trams" of types corresponding to the 0-1-string encoding of the types of the trams at the top positions of the stacks in the 0-DTDP-instance. Once again, the first 0 of the head part is always at the top-most position.

We prove that there is an assignment without shunting for the 0-DTDP instance if and only if there is an assignment without shunting for the 2-type 0-DTDP instance constructed above.

"⇒": In the first step, we assume that the instance of 0-DTDP can be solved with answer "yes". Hence, there is a shunting-free assignment of trams to departures. We assume that such an assignment is given by $\pi_Y(\tilde{d}_i) = p(i)$ for all $\tilde{d}_i \in \tilde{\mathcal{D}}$.

To each departure in the 0-DTDP instance corresponds a sequence of departures with types 0 or 1 in the 2-type 0-DTDP-instance. We obtain a shunting-free assignment for the 2-type 0-DTDP-instance in the following way: The consecutive departures whose types correspond to the whole string encoding the type of departure $\tilde{d}_i \in \tilde{\mathcal{D}}$ are assigned to the "trams" which are stored in a stack $\mathcal{P}_r$, $1 \leq r \leq R$, and whose types correspond to the whole string encoding the type of the tram at position $p(i)$ in the 0-DTDP-instance.

"⇐": We assume that we know a shunting-free departure for the constructed instance I of the 2-type 0-DTDP. For each departure, we show that such an assignment satisfies the following property. The sequence of departures in I whose types correspond to the whole string encoding the type $\tau \in \tilde{\mathcal{T}}$ of a departure in $\tilde{\mathcal{D}}$ is assigned to the "trams" which are stored in a stack $\mathcal{P}_r$, $1 \leq r \leq R$, and whose types correspond to the whole string encoding the type of a tram of the same type τ. This results in a shunting-free assignment for 0-DTDP.

We assume that the departure sequence $\tilde{\mathcal{D}}$ of the 0-DTDP-instance starts with a departure of type τ_i, $1 \leq i \leq \tilde{T}$. Then, the departure sequence $\mathcal{D}$ of the 2-type 0-DTDP-instance starts with a sequence of types [0][0]001, followed by the types corresponding to the code part of τ_i and by the types 0[1]1.

Before we can assign a "tram" of type 1 in the 2-type 0-DTDP instance to the $(2\lceil\log_2(\tilde{T})\rceil + 3)$-th departure of $\mathcal{D}$ without shunting, we have to assign $2\lceil\log_2(\tilde{T})\rceil + 2$ "trams" of type 0 from exactly one stack to the first $2\lceil\log_2(\tilde{T})\rceil + 2$ departures of $\mathcal{D}$. Then, we assign the "tram" of type 1 to the departure of type 1. We denote this stack as the **current stack**.

The next $\lceil\log_2(\tilde{T})\rceil$ departures of $\mathcal{D}$ correspond to the code part of τ_i. Since at the top of each stack, except of the current stack, there are $2\lceil\log_2(\tilde{T})\rceil + 2$ "trams" of type 0, we observe the following. First, we have to serve these departures from the same stack as before, i.e., from the current stack. Secondly, we must have chosen the current stack in such a way that it contains the sequence of "trams" whose types correspond to the 0-1-string encoding type τ_i. By $(*)$ (implying the existence of a type-preserving assignment) and by construction, such a stack exists.

For the next $\lceil\log_2(\tilde{T})\rceil + 2$ departures, we apply a similar argumentation. The second of these departures is of type 1. At the top of each stack, except of the current stack, there are still $2\lceil\log_2(\tilde{T})\rceil + 2$ "trams" of type 0. Therefore, these departures are served from the current stack by the "trams" whose types correspond to the tail of the 0-1-string encoding type τ_i.

We obtain that the first $4\lceil\log_2(\tilde{T})\rceil + 5$ departures of $\mathcal{D}$ (whose types correspond to the type of the first departure in $\tilde{\mathcal{D}}$) are served by the "trams" stored in exactly one stack. These "trams" correspond to a tram (of the same type encoded as described by a 0-1-string) in the 0-DTDP-instance.

For the next $4\lceil \log_2(\tilde{T}) \rceil + 5$ departures of $\mathcal{D}$ and step by step for all remaining departures, we continue in the same way as for the first departures. In each step, the "trams" with types corresponding the encoding of a tram type in the 0-DTDP-instance are taken from the same stack and assigned to the departures whose types correspond to the encoding of a departure type of $\tilde{\mathcal{D}}$.

In the assignment for the 0-DTDP instance constructed in this way, we assign to each departure of $\tilde{\mathcal{D}}$ a tram whose encoded type corresponds to the type sequence of the "trams" in the assignment for the 2-type 0-DTDP-instance. Finally, we obtain a shunting-free assignment for the 0-DTDP-instance. Consequently, the instance of 0-DTDP is solvable without shunting if and only if the corresponding instance of the 2-type 0-DTDP is solvable without shunting. This completes the proof. ❑

3.4.1 The Departure Problem with a Fixed Number of Stacks

In the following, we examine the departure problem (DTDP) for a depot with a fixed number R of stacks. We present a dynamic programming approach which solves 0-DTDP in polynomial time. This dynamic programming approach is polynomial in the number of departures but exponential in the number of stacks.

Given the assignment π_X that maps the N arriving trams to the N stack positions, we seek for an assignment π_Y that assigns the N departures to the trams stored at the stack positions. This assignment π_Y has to satisfy that

- to each departure, a tram of suitable type is assigned and
- each tram can leave the depot without shunting.

Such an assignment π_Y does not necessarily exist. If we assume that such an assignment exists, the only possibility to serve the first departure is to assign a tram (of suitable type) which is standing at a top position of a stack. The main idea of the dynamic programming approach can be described as follows. Since there are R stacks, we have at most R possibilities to choose a tram of suitable type. Let r_1 be the stack out of which the tram is taken to serve the first departure. For the second departure, we have again at most R possibilities for assigning a tram. This tram can be chosen out of the $R-1$ trams standing at the top positions in the stacks except of stack r_1. If we assume that stack r_1 consists of at least two positions, we can also decide to assign the tram standing at position $P_{r_1} - 1$ in stack r_1 if this tram is of suitable type. Note that the size (or capacity) of a stack r is denoted by P_r. We number the positions of each stack from 1 to P_r so that P_r denotes the top position of stack r.

Next, we formalize this procedure by a dynamic programming approach. We refer to [DL77, Smi91] for a comprehensive introduction to dynamic programming.

Definition 3.4.22: The **state space** $\mathcal{S}$ is given by

$$\mathcal{S} = \prod_{r=1}^{R} \{0, 1, \dots, P_r\}.$$

For each state $s = (s_1, s_2, \dots, s_R)$, we define $k(s) := \sum_{r=1}^{R} s_r$.

A state s corresponds to the situation where s_r trams are taken out of stack r without shunting. The $k(s)$ trams taken from the stacks serve the first $k(s)$ departures.

A state $s' = (s_1, \dots, s_r + 1, \dots, s_R)$ is said to **admit a shunting-free assignment** for the departures $d_1, \dots, d_{k(s)+1}$ if and only if departure $d_{k(s)+1}$ can be served by a tram of suitable type from position $P_r - s_r - 1$ of stack r, $1 \leq r \leq R$ and $s_r > 1$. This assignment is possible without shunting. Additionally, the state $s = (s_1, \dots, s_R)$ has to admit a shunting-free assignment for the departures $d_1, \dots, d_{k(s)}$.

The **boundary condition** is given by the state $s^0 = (0, 0, \dots, 0)$ which corresponds to the empty assignment where no departure has been assigned to a tram. For this empty assignment, no shunting movement is required so that s^0 is said to admit a shunting-free assignment. State $s^N = (P_1, P_2, \dots, P_R)$ represents the empty depot after the last departure is served. In this case, all trams have left the depot and all departures are served. ❑

The starting point for the dynamic program is state $s^0 = (0, \dots, 0)$ which represents the depot before the first departure is served. Starting from s^0, we search for a state s' which differs from s^0 in exactly one component s'_{r_1} where $s'_{r_1} = s_{r_1} + 1$. If s' admit a shunting-free assignment for d_1, we keep s' as a possible candidate for the first assignment of π_Y. We continue with all states s' that admit a shunting-free assignment for d_1. For all these states s', we search for a state s'' that differs from s' in exactly one component $s''_{r_2} = s'_{r_2} + 1$. Once again, we keep all the states s'' that admit a shunting-free assignment for d_2 (and d_1). We proceed iteratively until either state s^N is shown to admit a shunting-free assignment or for a departure $d_j \in \mathcal{D}$ $(j < N)$ no state $s \in \mathcal{S}$ exists that admits a shunting-free assignment for $d_1, \dots, d_j$.

Using the above procedure, we walk through the state space $\mathcal{S}$ for instance by breath-first-search (or depth-first-search). If we reach the state s^N and s^N admits a shunting-free assignment for $\mathcal{D}$, π_Y corresponds to a direct path from s^0 to s^N. Let $s^0, s^1, s^2, \dots, s^N$ be such a path where each state s^i $(1 \leq i \leq N)$ admits a shunting-free assignment. Then, π_Y assigns to departure d_i the tram stored in the stack r at position $P_r - s^i_r + 1$, where stack r is given by the component in which s^i and s^{i-1} differ.

The **principle of optimality** (cf. [DL77]) holds, since the order in which the trams are taken out of the different stacks is not relevant for the condition for a state to admit a shunting-free assignment. The best path from s^0 to s^N satisfies

the property that, independent of the first assignment, the remaining path to s^N starting from the subsequent state s^1 is the best path from s^1 to s^N. The same holds, step by step, for all states s^i.

Each stack size P_r is bounded from above by N, the number of trams stored in the depot. Consequently, $P_r = O(N)$ for all $1 \leq r \leq R$ and $\mid \mathcal{S} \mid = O(N^R)$. Since R is fixed, the number of states is polynomially bounded by the size of the input. For fixed R, we can apply a straight-forward search to compute the solution of 0-DTDP in polynomial time, for instance, by applying breadth-first-search starting from s^0. We achieve the following result.

Theorem 3.4.23: If the number of stacks is fixed, 0-DTDP is solvable in polynomial time.

This theorem has also been shown in [BBH$^+$98]. The dynamic programming approach can be extended to depots of N trams and departure sequences of $M < N$ departures. In this situation, we start from s^0 and try to find a path from s^0 to a state s with $k(s) = M$ that admits a shunting-free assignment. Moreover, in the same way we can determine the longest subsequence of $\mathcal{D}$ starting with d_1 that can be served without shunting. The length of this subsequence corresponds to the state s^* for which $k(s^*)$ is maximal among all states in $\mathcal{S}$ admitting a shunting-free assignment.

Unfortunately, applying an adapted dynamic program to the minimization problem DTDP does not lead to a polynomial algorithm. For the minimization problem, we must take into account the path that leads us to a state s. Hence, we have to consider all possible situations where trams of suitable type can be assigned causing some shunting movements. Even for a fixed number of stacks, this results in exponentially many states. However, this might be a promising approach to solve some instances in order to evaluate the performance of heuristics for DTDP. This approach corresponds to the enumeration algorithm BB of Chapter 5 (cf. Definition 5.3.1). We will go into more details in Chapter 5 where we will present some computational results for DTDP.

3.4.2 The Departure Problem for 2-Stacks

In the proof of Theorem 3.4.19, we have given a reduction from 3DM to the departure problem DTDP. Based on an arbitrary instance of 3DM, we constructed an instance of DTDP consisting of stacks of length 3. Therefore, DTDP is $\mathcal{NP}$-complete even if we restrict ourselves to instances with fixed stack length equal to 3. Next, we examine the case in which we restrict ourselves to stacks of length 2. We show that DTDP is solvable in polynomial-time for instances consisting of only two types.

Definition 3.4.24: We introduce the following restricted version of the departure problem DTDP.

Instance: Given

- a set of $N = 2R$ trams arranged in R stacks with two positions p_{2r-1} and p_{2r} where p_{2r} denotes the top position, $1 \leq r \leq R$,
- an assignment $\pi_X : \mathcal{A} \to \mathcal{P}$ that defines the arrangement of trams in the stacks where $\mathcal{A} = \{a_1, \ldots, a_N\}$ denotes the set of trams and $\mathcal{P}$ denotes the set of stack positions,
- a set of types $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$
- a set of departures $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ ordered by their indices, i.e., $d_i < d_j :\Leftrightarrow i < j$,
- and a mapping $t : \mathcal{A} \cup \mathcal{D} \to \mathcal{T}$.

Question: Is there an assignment $\pi_Y : \mathcal{D} \to \mathcal{P}$ and a positive integer K such that

1. $t(\pi_X^{-1}(p)) = t(\pi_Y^{-1}(p))$ for all $p \in \mathcal{P}$ and
2. $\mid \{r \mid \pi_Y^{-1}(p_{2r-1}) < \pi_Y^{-1}(p_{2r}), 1 \leq r \leq R\} \mid \leq K$.

We denote by **2DTDP** the decision problem is to whether or not there is an assignment π_Y for a given value of K. The corresponding minimization problem in which we are interested in the minimum number K is denoted by **min-2DTDP**. ❑

The first property means whether or not it is possible to assign a tram of suitable type to every departure of $\mathcal{D}$. This can be checked easily in advance. An assignment satisfying the first property exists if and only if, for each type, there are exactly as many trams as departures of the same type. In this section, we assume that the underlying instance admits an assignment satisfying the first property. In the second property, we count the number of stacks where the tram at the bottom position is assigned to an earlier departure than the tram at the top position. Hence, the number of such stacks is equal to the number of shunting movements required. In the following, we examine if min-2DTDP is solvable in polynomial-time.

The N trams are stored in R stacks each of size 2. We identify the trams by the positions $p \in \mathcal{P}$ at which they are stored in the stacks and distinguish the trams by their positions in the stacks. We define $\mathfrak{T} := \{a \in \mathcal{A} \mid \pi_X(a) = p_{2r}, 1 \leq r \leq R\}$ to be the set of trams stored at the stacks' top positions. Analogously, by $\mathfrak{B} := \{a \in \mathcal{A} \mid \pi_X(a) = p_{2r-1}, 1 \leq r \leq R\}$, we denote the set of trams that are stored at the bottom positions. The number of trams in $\mathfrak{T}$ of type τ is denoted by $n(\tau)$, $\tau \in \mathcal{T}$.

Proposition 3.4.25: There is an optimal solution of min-2DTDP in which only trams of $\mathfrak{T}$ are assigned to the first $n(\tau)$ departures of type τ for all types $\tau \in \mathcal{T}$.

Proof: We assume that there is an optimal solution π_Y^* of min-2DTDP which assigns at least one tram of $\mathfrak{B}$ to a departure of $\mathcal{D}$ having a smaller index than the departure to which the last tram of $\mathfrak{T}$ of the same type is assigned.

Hence, there are at least two stacks r and s so that

$$\pi_Y^*(p_{2r-1}) = d_i \text{ and } \pi_Y^*(p_{2s}) = d_j, \quad t(d_i) = t(d_j), i < j. \tag{3.4.1}$$

Note that the optimality of π_Y^* implies that $r \neq s$. If $r = s$, we could compute a better assignment than π_Y^* by assigning p_{2r-1} to d_j and $p_{2s} = p_{2r}$ to d_i.

Next, we present an assignment $\tilde{\pi}_Y$ which does not require more shunting movements than π_Y^*.

We define $\tilde{\pi}_Y(p) := \pi_Y^*(p)$ for all $p \in \mathcal{P}\backslash\{p_{2r-1}, p_{2s}\}$, $\tilde{\pi}_Y(p_{2r-1}) = d_j$, and $\tilde{\pi}_Y(p_{2s}) = d_i$.

Since $i < j$, the tram at the bottom position is now assigned to a later departure and the tram at the top position is assigned to an earlier departure. Hence, no additional shunting movement is needed. It follows that $\tilde{\pi}_Y$ is also an optimal assignment.

This procedure is iterated as long as there are two stacks for which (3.4.1) holds. If (3.4.1) is not satisfied by two stacks, we have derived an optimal solution for which the proposition holds. ❑

For each departure $d \in \mathcal{D}$, we can now determine if an element of $\mathfrak{T}$ or an element of $\mathfrak{B}$ is assigned to it. We partition $\mathcal{D}$ into two sets $\mathcal{D}_{\mathfrak{T}}$ and $\mathcal{D}_{\mathfrak{B}}$. A departure d belongs to $\mathcal{D}_{\mathfrak{T}}$ if a tram of $\mathfrak{T}$ is assigned to it. Otherwise, it belongs to $\mathcal{D}_{\mathfrak{B}}$. $\mathcal{D}_{\mathfrak{T}}$ and $\mathcal{D}_{\mathfrak{B}}$ contain exactly R elements each.

If there are stacks in which two trams of the same type are stored, we can fix the assignment π_Y for this stack in the following way:

Lemma 3.4.26: Let $\mathcal{P}_r = \{p_{2r-1}, p_{2r}\}$ be a stack with $t(\pi_X^{-1}(p_{2r-1})) = t(\pi_X^{-1}(p_{2r})) = \tau$. There is an optimal assignment which assigns the tram at position p_{2r} to the (last) departure $d \in \mathcal{D}_{\mathfrak{T}}$ of type τ having maximum index and the tram at position p_{2r-1} to the (first) departure $d \in \mathcal{D}_{\mathfrak{B}}$ of type τ having minimum index.

Proof: We assume that π_Y is an optimal assignment for min-2DTDP. Additionally, we assume that π_Y is defined in accordance with Proposition 3.4.25.

We denote by d_k the departure in $\mathcal{D}_{\mathfrak{T}}$ of type τ having maximum index and by d_h the departure in $\mathcal{D}_{\mathfrak{B}}$ of type τ having minimum index. Due to Proposition 3.4.25, $k < h$.

For p_{2r} and p_{2r-1}, let π_Y be given by,

$$\begin{aligned}\pi_Y^{-1}(p_{2r}) &= d_i,\\ \pi_Y^{-1}(p_{2r-1}) &= d_j.\end{aligned}$$

Owing to Proposition 3.4.25, $i \leq k$. If $i < k$, then there is a position p_q, $q \neq 2r$, with $\pi_Y(d_k) = p_q$. By Proposition 3.4.25, p_q is in $\mathfrak{T}$. We interchange the assignment for d_i and d_k and obtain the following assignment π_Y^*,

$$\begin{aligned}\pi_Y^*(d_k) &= p_{2r},\\ \pi_Y^*(d_i) &= p_q.\end{aligned}$$

and $\pi_Y(p) = \pi_Y^*$ for all $p \in \mathcal{P}, p \neq p_{2r}, p_q$.

Owing to Proposition 3.4.25, $j \geq h$. If $j > h$, then there is position p_l, $l \neq 2r-1$, with $\pi(d_h) = p_l$. By Proposition 3.4.25, p_l is in $\mathfrak{B}$. Once again, we switch the assignment for d_j and d_h and obtain the following assignment π_Y^*,

$$\begin{aligned}\pi_Y^*(d_h) &= p_{2r-1},\\ \pi_Y^*(d_j) &= p_l.\end{aligned}$$

and $\pi_Y^*(p) = \pi_Y(p)$ for all $p \in \mathcal{P}, p \neq p_{2r-1}, p_l$.

π_Y^* does not require more shunting movements than π_Y because

- $i < k$ so that the tram at p_q is removed by π_Y^* before this tram is removed by π_Y,
- $j > h$ so that the tram at p_l is removed by π_Y^* later than it is removed by π_Y, and
- the assignment of the trams at p_{2r} and p_{2r-1} does not require shunting.

❑

In the following, without loss of generality we assume that the 2DTDP-instances to be considered do not contain stacks with trams of the same type. Every instance containing such a stack can be reduced to an instance without such stacks by fixing the assignment of trams to departures as described in Lemma 3.4.26.

Next, we define a complete, bipartite, weighted graph in which we search for a perfect matching of minimum weight. The minimum weight of a perfect matching will be shown to be a lower bound on the number of shunting movements for 2DTDP. For the case that the instance consists of trams and departures of only two types this lower bound is tight; i.e., the minimum weight of a perfect matching is equal to the minimum number of shunting movements required for the DTDP-instance (cf. Theorem 3.4.28).

The weighted bipartite graph G is given by $G := (\mathcal{D}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{B}}, \mathcal{D}_{\mathfrak{T}} \times \mathcal{D}_{\mathfrak{B}})$. The weight of an edge $(d_i, d_j) \in \mathcal{D}_{\mathfrak{T}} \times \mathcal{D}_{\mathfrak{B}}$ is defined as follows. If there is a stack

containing a tram of type $t(d_i)$ at the top and a tram of type $t(d_j)$ at the bottom position, then

$$w(d_i, d_j) = \begin{cases} 0 & \text{if } i < j, \\ 1 & \text{if } i > j. \end{cases}$$

Otherwise, $w(d_i, d_j)$ is set to $R + 1$ so that this edge will never be chosen in a minimum weighted perfect matching.

Given a feasible assignment of trams to departures (or departures to stack positions), we construct a perfect matching as follows. Without loss of generality, we assume that this assignment is given in accordance with Proposition 3.4.25. If $d_i \in \mathcal{D}_{\mathfrak{T}}$ and $d_j \in \mathcal{D}_{\mathfrak{B}}$ are assigned to the positions in the same stack (where d_i is assigned to the top position and d_j is assigned to the bottom position), then this assignment corresponds to the edge (d_i, d_j) in the matching. This matching is perfect because the assignment of departures to stack positions is a bijective mapping between $\mathcal{D}$ and $\mathcal{P}$ and implies a bijective mapping between $\mathcal{D}_{\mathfrak{T}}$ and $\mathcal{D}_{\mathfrak{B}}$. The edge weight is equal to 1 if and only if $i > j$, i.e., if and only if the tram's departure requires shunting. Hence, to each feasible assignment of trams to departures corresponds a perfect matching in G of weight equal to the number of shunting movements required by the assignment.

Theorem 3.4.27: The minimum weight of a perfect matching in the bipartite graph G introduced above is a lower bound on the number of shunting movements for the corresponding 2DTDP-instance.

Proof: The theorem follows immediately by observing that every feasible solution of the 2DTDP instance corresponds to a perfect matching in G having a weight equal to the number of shunting movements required by the solution. ❑

The minimum weight of a perfect matching in G is only a lower bound because there may be perfect matchings in G that do not correspond to a feasible assignment of trams to departures.

Next, we consider the departure problem for stacks with a fixed number of two positions and with trams having only two types τ_1 or τ_2.

Theorem 3.4.28: If we restrict ourselves on instances with stacks with two positions and trams of only two types, then the departure problem min-2DTDP is polynomially solvable.

Proof: In accordance with Proposition 3.4.25, we determine $\mathcal{D}_{\mathfrak{T}}$ and $\mathcal{D}_{\mathfrak{B}}$. The set $\mathcal{D}_{\mathfrak{T}}$ consists of $n(\tau_1)$ departures of type τ_1 and of $n(\tau_2)$ departures of type τ_2. These departures are served by the trams stored at the stacks' top positions. By Lemma 3.4.26, we can assume without loss of generality that, in each stack, either a tram of type τ_1 is on top of a tram having type τ_2, or a tram of type τ_2 is on top of a tram having type τ_1.

In the case that the largest index of a departure in $\mathcal{D}_{\mathfrak{T}}$ is smaller than the smallest index of a departure in $\mathcal{D}_{\mathfrak{B}}$, the departure sequence $\mathcal{D}$ can be served without shunting. Otherwise, there are a number of departures in $\mathcal{D}_{\mathfrak{B}}$ having a smaller index than the last departure in $\mathcal{D}_{\mathfrak{T}}$.

We consider a perfect matching of minimum weight in the graph G introduced above. Since G is a complete bipartite graph, such a matching exists. As shown above, each solution of the 2DTDP-instance considered in this theorem corresponds to a perfect matching in G having a weight equal to the number of shunting movements required by the solution.

Since there is a perfect matching in G corresponding to a feasible assignment and having a weight equal to the number of shunting movements required by this assignment, the minimum weight of a perfect matching is not greater than R. To each edge (d_i, d_j) of a perfect matching in G, there corresponds a stack in the 2DTDP-instance. By assumption (cf. Lemma 3.4.1), in all stacks containing a tram of type $t(d_i)$ at the top there is a tram of type $t(d_j) \neq t(d_i)$ at the bottom position. Consequently, there are as many departures in $\mathcal{D}_{\mathfrak{T}}$ having type $t(d_i)$ as stacks with a tram of type $t(d_i)$ at the top position and a tram of type $t(d_j)$ at the bottom position. There are $n(t(d_i))$ stacks with this property.

Given a perfect matching, we construct a feasible assignment for the 2DTDP-instance as follows. Initially, we mark all stacks to be "unassigned". For each matching edge (d_i, d_j), we arbitrarily choose an "unassigned" stack containing a tram of type $t(d_i)$ at the top and a tram of type $t(d_j)$ at the bottom and mark this stack to be "assigned". We assign d_i to the top position of this stack and d_j to the bottom position. We repeat this procedure for each matching edge.

The weight of the assignment constructed in this way is equal to the weight of the perfect matching since the edge weight of (d_i, d_j) is equal to one if and only if $i > j$ which implies that the assigned trams have to be shunted at departure. (Note that in a minimum weighted perfect matching no edge having a weight of $R + 1$ is chosen.)

Hence, the minimum number of shunting movements is equal to the minimum weight of a perfect matching in G which can be determined in polynomial time [Tom71]. ❑

The correctness of the above theorem requires that the number of stacks containing a tram of type τ_i on top of a tram of type τ_j $(i \neq j)$ is equal to the number of departures of type τ_i in $\mathcal{D}_{\mathfrak{T}}$. Even if we restrict ourselves on 2DTDP-instances with three types τ_1, τ_2, τ_3, this property does not hold. For an arbitrary number of types, the minimum weighted matching only implies a lower bound on the number of shunting movements. In this case, it may be possible that we assign more departures of type τ_i in $\mathcal{D}_{\mathfrak{T}}$ to departures of type τ_j in $\mathcal{D}_{\mathfrak{B}}$ than there are stacks containing a tram of type τ_i on top of a tram of type τ_j. However, this lower bound derived by determining a perfect matching of minimum weight in G can be improved by a careful edge deletion procedure.

Example 3.4.29: The following instance of 2DTDP with three types $\{\tau_1, \tau_2, \tau_3\}$ requires one shunting movement whereas the minimum weight of a perfect matching in the corresponding bipartite graph is zero.

The departure sequence $\mathcal{D} = \{d_1, d_2, \ldots, d_{12}\}$ is given by the following sequence of types starting with d_1:

$$(\tau_1, \tau_1, \tau_3, \tau_3, \tau_3, \tau_3, \tau_2, \tau_2, \tau_1, \tau_1, \tau_2, \tau_2)$$

The trams and the stacks are given in accordance with Figure 3.4.1.

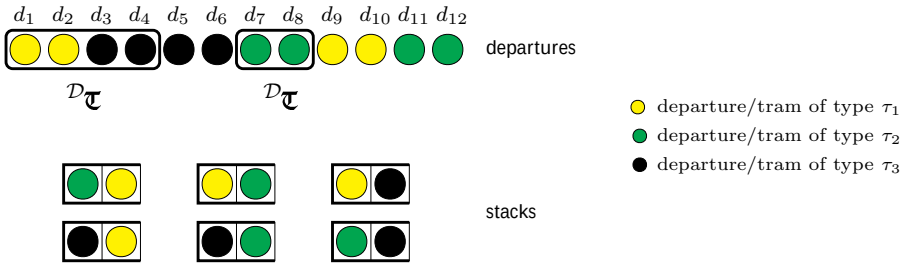

Figure 3.4.1: The 2DTDP-instance of Example 3.4.29.

Since all trams of type τ_3 must leave before the trams of type τ_2, at least one shunting movement is necessary for the trams in the stack containing a tram of type τ_2 on top of a tram of type τ_3. We obtain that $\mathcal{D}_{\mathfrak{T}} = \{d_1, d_2, d_3, d_4, d_7, d_8\}$ and $\mathcal{D}_{\mathfrak{Z}} = \{d_5, d_6, d_9, d_{10}, d_{11}, d_{12}\}$.

The corresponding matrix $W = \big(w(d_i, d_j)\big)$ of the edge weights of G is given as follows ($d_i \in \mathcal{D}_{\mathfrak{T}}, d_j \in \mathcal{D}_{\mathfrak{Z}}$):

$$W = \begin{pmatrix} 0 & 0 & R+1 & R+1 & 0 & 0 \\ 0 & 0 & R+1 & R+1 & 0 & 0 \\ R+1 & R+1 & 0 & 0 & 0 & 0 \\ R+1 & R+1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & R+1 & R+1 \\ 1 & 1 & 0 & 0 & R+1 & R+1 \end{pmatrix}.$$

A perfect matching of minimum weight in G contains, for instance, the following edges:

$$(d_1, d_5), (d_2, d_6), (d_3, d_{11}), (d_4, d_{12}), (d_7, d_9), \text{ and } (d_8, d_{10}).$$

This perfect matching has a weight of zero.

The **edge deletion procedure** works as follows. For each departure $d_i \in \mathcal{D}_{\mathfrak{T}}$ and each edge (d_i, d_j) of weight 0 adjacent to d_i, we successively check whether or not a stack exists that may correspond to this edge. We first count the number

of departures of type $\tau = t(d_i)$ in the set $\{d_k \in \mathcal{D}_{\mathfrak{T}} \mid k < i\}$. We then count the number of departures of type $\bar{\tau}$ in the set $\{d_l \in \mathcal{D}_{\mathfrak{B}} \mid l \leq j\}$ for which a stack with a tram of type τ on top of a tram of type $\bar{\tau}$ exists. If the first number is not smaller than the second number, then the weight of (d_i, d_j) is set to $R+1$. In this case, the departure d_j can be combined with a departure in $\{d_k \in \mathcal{D}_{\mathfrak{T}} \mid k < i\}$, i.e., both departures are assigned to the same stack. Since the cardinality of $\mathcal{D}_{\mathfrak{T}}$ and $\mathcal{D}_{\mathfrak{B}}$ is equal to R and there are at most R^2 edges in G where R denotes the number of stacks, this procedure runs in polynomial time.

In Example 3.4.29, the edges (d_2, d_5) and (d_2, d_6) are deleted, since there is only one stack containing a tram of type τ_1 on top of a tram of type τ_3. After having also deleted (d_4, d_9), (d_4, d_{10}), (d_8, d_9), and (d_8, d_{10}), we obtain the modified weight matrix W' given by

$$W' = \begin{pmatrix} 0 & 0 & R+1 & R+1 & 0 & 0 \\ R+1 & R+1 & R+1 & R+1 & 0 & 0 \\ R+1 & R+1 & 0 & 0 & 0 & 0 \\ R+1 & R+1 & R+1 & R+1 & 0 & 0 \\ 1 & 1 & 0 & 0 & R+1 & R+1 \\ 1 & 1 & R+1 & R+1 & R+1 & R+1 \end{pmatrix}.$$

For this weight matrix, the weight of a minimum weight perfect matching in G is equal to one. However, there are some more complicated counterexamples which prove that the lower bound derived by determining a minimum weight perfect matching in G (in the way as described above) is not tight.

3.5 The Departure Type Mismatch Problem

Analogously to Definition 3.3.15, we define the type mismatch problem restricted to the departures. In this problem, we are allowed to assign the stored trams to departures of different type but without causing shunting movements.

Definition 3.5.30: The **Departure Type Mismatch Problem (DTMP)** is given by the following instance and the following question:

Instance:
- a set $\mathcal{A} = \{a_1, \ldots, a_N\}$ of arriving trams
- a set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of departures
- a set of N depot positions $\mathcal{P}$ located in R stacks of sizes P_r, $1 \leq r \leq R$, where the P_r positions in stack $\mathcal{P}_r \subset \mathcal{P}$ are numbered consecutively from the bottom to the top and denoted by $\sum_{i=1}^{r-1} P_i + 1, \ldots, \sum_{i=1}^{r} P_i$,
- a mapping $t : \mathcal{A} \cup \mathcal{D} \to \{\tau_1, \ldots, \tau_T\}$ which assigns to each element of $\mathcal{A}$ and $\mathcal{D}$ a type in $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$.
- an assignment $\pi_X : \mathcal{A} \to \mathcal{P} = \bigcup_{r=1}^{R} \mathcal{P}_r$ (and the corresponding assignment matrix X) which maps the N trams of $\mathcal{A}$ to the N depot positions.

Question: Is there an assignment $\pi_Y : \mathcal{D} \to \mathcal{P}$ represented by the assignment matrix Y satisfying

$(*)$ $SMD = \bigcup_{r=1}^{R} \{(d_j, d_k) \in \mathcal{D} \times \mathcal{D} \mid y_{jq} = 1, y_{kl} = 1, j < k, p_q, p_l \in \mathcal{P}_r, \text{and } q < l\} = \emptyset$ (cf. Definition 3.4.18)

and minimizing the number of positions $p_q \in \mathcal{P}$ for which

$(**)$ $x_{iq} = 1 \quad \wedge \quad y_{jq} = 1 \quad \wedge \quad t(a_i) \neq t(d_j)$,

where $p_q \in \mathcal{P}, a_i \in \mathcal{A}, d_j \in \mathcal{D}$.

If $(**)$ holds for a position $p \in \mathcal{P}$, we say that there is a **type mismatch** at position p. In this case, a tram (stored at position p) is assigned to a departure that requires a type different from the tram's type.

Analogously to Definition 3.2.5 and Definition 3.3.15, we define the corresponding decision problem to DTMP as the problem of finding feasible assignments for which SMD is empty and the number of positions satisfying $(**)$ is bounded from above by K. The decision problem where we seek for a feasible assignment with $K = 0$ is denoted by **0-DTMP**. ❑

Since 0-DTDP and 0-DTMP are equivalent problems, we obtain the following theorem.

Theorem 3.5.31: 0-DTMP is $\mathcal{NP}$-complete. DTMP is $\mathcal{NP}$-hard.

Proof: This result is an immediate consequence of Theorem 3.4.19 and Corollary 3.4.20. ❑

In Section 3.4.1, we introduced a dynamic programming approach for 0-DTDP

for a fixed number R of stacks. By this dynamic programming approach, we can decide in polynomial time whether or not there is a shunting-free and type preserving assignment for the given 0-DTDP-instance and fixed R. Since 0-DTDP and 0-DTMP are equivalent, the same holds for 0-DTMP.

Corollary 3.5.32: 0-DTMP is decidable in polynomial time if we restrict ourselves to instances with a fixed number of stacks.

In the following, we examine the time complexity of DTMP. By π_X, we denote the given assignment of trams to depot positions. Given π_X, the problem is to compute a shunting-free assignment π_Y which minimizes the number of type mismatches. For this problem, we extend the above dynamic programming approach.

The trams of specified type are stored in R stacks. Each stack contains P_r trams. Analogously to Definition 3.4.22, we define the following dynamic programming approach:

Definition 3.5.33: For $\mathcal{P} = \bigcup_{r=1}^{R} \mathcal{P}_r$, $\mid \mathcal{P}_r \mid = P_r$, we define the *state space* $\mathcal{S}$ where

$$\mathcal{S} := \prod_{i=1}^{R} \{0, 1, \dots, P_r\}$$

To a state $s = (s_1, s_2, \dots, s_R) \in \mathcal{S}$, we identify the situation in which the topmost s_r trams of stack r $(1 \leq r \leq R)$ are assigned to the first $k(s) := \sum_{r=1}^{R} s_r$ departures of $\mathcal{D}$ in such a way that the resulting assignment of trams to departures does not require any shunting.

We denote the state $(0, 0, \dots, 0)$ by s^0. For each state $s \in \mathcal{S} \setminus \{s^0\}$, we define the set $F(s)$ of preceding states,

$$F(s) := \{s' \in \mathcal{S} \mid \forall\, r : s'_r \leq s_r \text{ and } k(s') = k(s) - 1\}.$$

By definition, the states in $F(s)$ differ from state s in only one component s_r by exactly 1. The corresponding position in this stack r is given by $q(s, r) := \sum_{i=1}^{r} P_i - s'_r$. The tram stored at this position p_q is supposed to be assigned to the next departure $d_{k(s)}$ of $\mathcal{D}$.

The state transition step is defined by

$$V(s) := \min\{V(s') + \delta(s, s') \mid s' \in F(s)\}, \quad s \in \mathcal{S}, s \neq s^0, \tag{3.5.2}$$

where

$$\delta(s, s') = \begin{cases} 1 & \text{if } t(\pi_X^{-1}(p_{q(s,r)})) \neq t(d_{k(s)}), \\ 0 & \text{otherwise.} \end{cases}.$$

The boundary condition is given by $V(s^0) = 0$.

Starting with $s^N := (P_1, P_2, \ldots, P_R)$, we define $\pi_Y : \mathcal{D} \to \mathcal{P}$ recursively by

$$\pi_Y(d_{k(s^i)}) = p_{q(s^i,r)}, \quad 1 \leq i \leq N,$$

where $s^i := \text{argmin}\,\{V(s') + \delta(s^{i+1}, s') \mid s' \in F(s^{i+1})\}$, $1 \leq i \leq N-1$, and r is the stack index in which s^i and s^{i+1} differ. ❑

By definition, the cardinality of $\mathcal{S}$ is bounded from above by N^R since there are N trams stored in the R stacks bounding the stack lengths by N.

Theorem 3.5.34:

1. The optimal solution value of DTMP is given by $V(s^N)$.
2. For fixed R, DTMP is solvable in polynomial-time.

Proof: First, we prove that the optimal solution is given by $V(s^N)$. We show that the recursive formula (3.5.2) holds for every state $s \in \mathcal{S} \setminus \{s^0\}$.

For $k(s) = 1$, the recursive formula holds since $V(s^0) = 0$ and $\delta(s, s^0)$ is equal to one if and only if the first departure d_1 is assigned to a position at which a tram of non-matching type is stored. Otherwise, $\delta(s, s^0)$ is zero.

Next, we consider a shunting-free assignment π_Y for the first $k(s) - 1$ departures of $\mathcal{D}$ and assume that (3.5.2) holds for every state $s' \in \mathcal{S}$ with $k(s') = k(s)-1$. For all $s' \in F(s)$, $V(s) \leq V(s')+\delta(s, s')$, since extending π_Y by assigning departure $d_{k(s)}$ to the corresponding position $q(s, r)$ may lead to one additional type mismatch resulting in $V(s') + \delta(s, s')$ type mismatches.

For some stack r and some state $s' \in F(s)$, the departure $d_{k(s)}$ is assigned to the tram at the corresponding position $q(s, r)$. This assignment requires $\delta(s, s')$ type mismatches, i.e., zero or one. Hence the assignment of $d_1, \ldots, d_{k(s)-1}$ requires $V(s) - \delta(s, s')$ type mismatches. Consequently $V(s) - \delta(s, s') \geq V(s')$, which proves (3.5.2).

As a result, $V(s^N)$ corresponds to a shunting-free assignment π_Y that assigns the departures of $\mathcal{D}$ to the trams at the stack positions. By (3.5.2), $V(s^N)$ is the minimum value of an assignment extending the partial assignment of the first $N - 1$ departures to the N departures of $\mathcal{D}$. Since (3.5.2) holds for every state $s \in \mathcal{S} \setminus \{s^0\}$, by induction, $V(s^N)$ denotes the minimum number of type mismatches required for the DTMP instance.

The optimal value $V(s^N)$ is either obtained by a recursive computation or starting from s^0 for instance by breadth-first-search. The cardinality of the state space $\mathcal{S}$ is bounded from above by $O(N^R)$ since the R stacks are at most of length N. Since breadth-first-search runs in time polynomial in the size of the search space, for fixed R the optimal solution can be determined in polynomial-time. ❑

Chapter 4

Models and Algorithms

In this chapter, we present mathematical formulations and models for the tram dispatch problem TDP (cf. Definition 3.2.5) and the type mismatch problem TMP (cf. Definition 3.2.5) introduced in the previous chapter. We give a quadratic binary program for TDP which is a combination of two quadratic assignment problems. For TMP, we present a binary linear program. Based on the observations for these problems, we introduce related models for the corresponding dispatch problems at departure, i.e., for DTMP and DTMP.

4.1 A Quadratic Program for TDP

In this section, we present a quadratic programming formulation for the tram dispatch problem (TDP).

An instance of TDP consists of

- a set of incoming trams $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$,
- a set of round trips (departures) $\mathcal{D} = \{d_1, d_2, \dots, d_M\}$ that have to be served,
- a set of (stack) positions $\mathcal{P} = \{p_1, p_2, \dots, p_P\}$ to which the arriving trams have to be assigned, and
- R stacks in which the positions are located.

Throughout the whole thesis, we identify each round trip by the corresponding departure of a tram. We assume that the arrivals and departures are linearly ordered with respect to their indices. The trams of $\mathcal{A}$ are ordered by their scheduled arrival time, i.e., the indices of the trams a_i, a_j in $\mathcal{A}$ correspond to this ordering where tram a_i arrives before tram a_j if and only if $i < j$. We assume that the same holds for the departures, i.e., departure $d_i \in \mathcal{D}$ has to be served before departure $d_j \in \mathcal{D}$ if and only if $i < j$.

According to Section 2.2, we have specified for each tram of $\mathcal{A}$ and each departure of $\mathcal{D}$ a type $\tau \in \mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_T\}$. We denote this definition of types by the function $t : \mathcal{A} \cup \mathcal{D} \rightarrow \mathcal{T}$.

The positions $p_1, p_2, \ldots, p_P$ are located in R stacks with sets of positions $\mathcal{P}_1, \ldots, \mathcal{P}_R$. The sets $\mathcal{P}_1, \ldots, \mathcal{P}_R$ define a partition of $\mathcal{P}$. Let P_r denote the number of positions in $\mathcal{P}_r$. Let $bottom_r = 1 + \sum_{i=1}^{r-1} P_i$ and $top_r = bottom_r + P_r - 1$, $1 \leq r \leq R$ where $bottom_r$ denotes the bottom and top_r denotes the top position of stack $\mathcal{P}_r$, $1 \leq r \leq R$. Position $p_i \in \mathcal{P}$ is in stack $\mathcal{P}_r$ if $bottom_r \leq i \leq top_r$.

An assignment $\pi_X : \mathcal{A} \rightarrow \mathcal{P}$ of the arriving trams to the positions is identified by an $(N \times P)$-dimensional 0-1-matrix $X = (x_{iq})$. If tram a_i is assigned to position p_q, i.e., $\pi_X(a_i) = p_q$, then $x_{iq} = 1$. Otherwise $x_{iq} = 0$. We call such a matrix **assignment matrix** (cf. Chapter 3).

The assignment $\pi_Y : \mathcal{D} \rightarrow \mathcal{P}$ of the departures to the positions is defined by the $(M \times P)$-dimensional 0-1-matrix $Y = (y_{jq})$. The departure d_j is said to be **served** by the tram at position p_q ($\pi_Y(d_j) = p_q$) if $y_{jq} = 1$. The case $y_{jq} = 0$ implies that departure d_j is not served by the tram standing at position p_q.

Such a pair of assignments π_X and π_Y is said to be **feasible for TDP** if the two following conditions are satisfied:

- The set of positions to which a departure is assigned must be covered by the set of positions to which an arriving tram is assigned. More formally, $\{p_q \in \mathcal{P} \mid y_{jq} = 1, d_j \in \mathcal{D}\} \subseteq \{p_q \in \mathcal{P} \mid x_{iq} = 1, a_i \in \mathcal{A}\}$
- For each position the type of the arriving tram and the type of the departure assigned to it must be identical.

We also refer to feasible assignments for TDP as being **type-preserving**. Since the definition of types is assumed to be made in accordance with Section 2.2, we assume that the second condition is satisfied. Note that the first condition implies $P \geq N \geq M$. In the following, we assume that $N = M = P$. (The other cases will be discussed in Section 4.1.4).

As a consequence of this assumption, if $x_{iq} = 1$, there must be an index j so that $y_{jq} = 1$. Additionally, the types of a_i and d_j must be identical. Therefore, the following condition must hold:

$$\big(x_{iq} = 1 \wedge y_{jq} = 1\big) \quad \Rightarrow \quad t(a_i) = t(d_j) \quad \text{for all } a_i \in \mathcal{A},\ d_j \in \mathcal{D}, \text{ and } p_q \in \mathcal{P}.$$

This means that the type of each tram has to be the same as required by the departure to which this tram is assigned.

Such pair of feasible assignments exists if and only if there are exactly as many arriving trams of a type $\tau \in \mathcal{T}$ as departures of this type τ (for any arbitrary type $\tau \in \mathcal{T}$). In the following, we assume this condition being satisfied.

Since to each departure a tram can only be assigned if it has arrived before, we introduce the following notations. To each departure, we assign a tram which is assumed to be already standing at a position in the depot. This tram is identified

by its standing position in the depot and, of course, by its arrival index i. We say that the tram a_i standing at position p_q is assigned to departure d_j if tram a_i and departure d_j are assigned to the same position p_q. In the tram dispatch problem, we assume that the first departure is scheduled after the last tram has arrived at the depot. Consequently, to each position only one tram and only one departure can be assigned.

A feasible assignment without rearranging the stacks does not necessarily exist. An instance of TDP may force the dispatcher to shunt some trams either while assigning the trams to the stacks or while the trams leave the depot to serve the round trips. Since shunting takes time and requires personnel, it should be reduced to a minimum. In Figure 4.1.1, we give an instance of TDP. If we assign tram a_1 to the top position p_3 of stack $\mathcal{P}_1$, the assignment of tram a_i $(i > 1)$ to position p_1 or p_2 implies that tram a_1 has to be moved, before a_i can enter the stack.

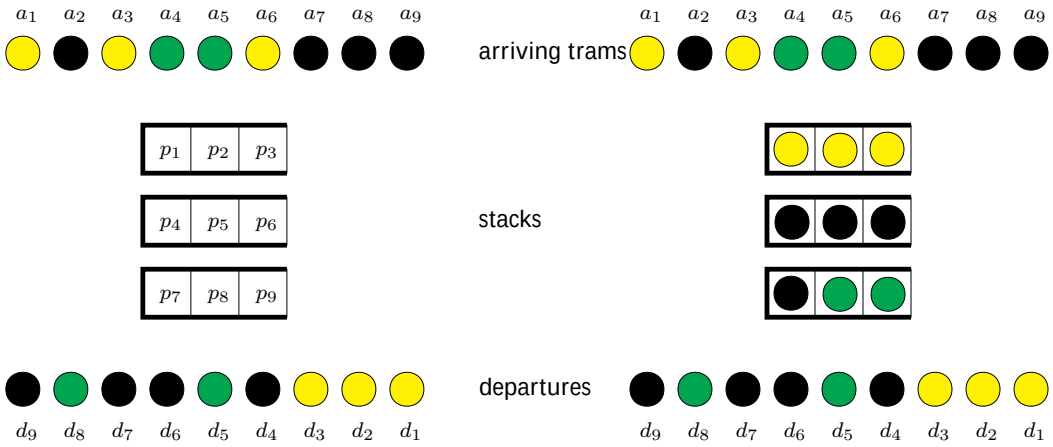

Figure 4.1.1: An instance of the TDP and a possible solution.

In Figure 4.1.1, shunting of trams is avoidable. For instance, the following assignments π_X and π_Y represented by the assignment matrices $X = (x_{iq})$ and $Y = (y_{iq})$ are possible without rearranging trams.

arrival	position	departure	x	y
a_1	p_1	d_3	$x_{11} = 1$	$y_{31} = 1$
a_2	p_7	d_9	$x_{27} = 1$	$y_{97} = 1$
a_3	p_2	d_2	$x_{32} = 1$	$y_{22} = 1$
a_4	p_8	d_8	$x_{48} = 1$	$y_{88} = 1$
a_5	p_9	d_5	$x_{59} = 1$	$y_{59} = 1$
a_6	p_3	d_1	$x_{63} = 1$	$y_{13} = 1$
a_7	p_4	d_7	$x_{74} = 1$	$y_{74} = 1$
a_8	p_5	d_6	$x_{85} = 1$	$y_{65} = 1$
a_9	p_6	d_4	$x_{96} = 1$	$y_{46} = 1$

Our goal is to minimize shunting, or in other words, to minimize the number of shunting movements. Shunting of two trams is necessary if these trams are placed in the same stack but in a different order than defined by the sequence of arrivals or departures. In this case, these two trams have to be rearranged in order to store the trams or to let them depart. Such a conflict between two trams is called a **shunting movement**. We count the number of shunting movements separately for the arrivals and the departures. The objective is to minimize the total number of shunting movements. The cost of assigning a tram to a position is defined as the number of shunting movements necessary for this assignment, i.e., the (minimal) number of shunting movements necessary to assign the tram in the order given by the assignment. Analogously, the cost of assigning a tram at a position to a departure is defined as the (minimal) number of shunting movements needed to let the tram leave the depot.

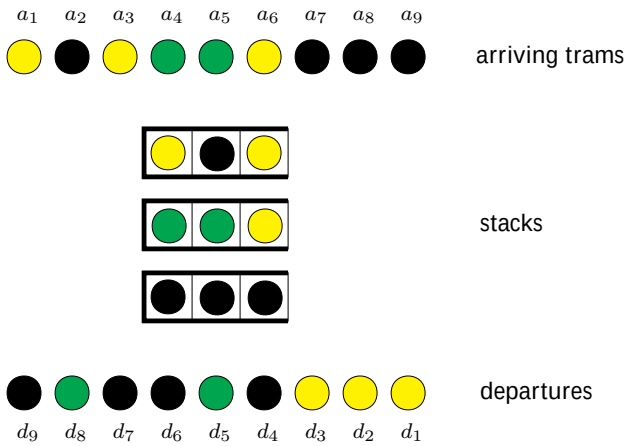

Figure 4.1.2: Another possible configuration of the storage yard.

For each arriving tram, the cost of assigning it to a position depends on the assignment of further trams to the same stack. In the following, we develop a quadratic program for the TDP. Analogously to the definition of the assignment

matrices, for each arriving tram $a_i \in \mathcal{A}$ and all positions $p_q \in \mathcal{P}$ we introduce binary variables x_{iq} that we interpret as follows.

Definition 4.1.1: For all arriving trams $a_i \in \mathcal{A}$ and all positions $p_q \in \mathcal{P}$, we define x_{iq} as follows:

$$x_{iq} := \begin{cases} 1 & \text{if tram } a_i \text{ is assigned to position } p_q \\ 0 & \text{otherwise} \end{cases} .$$

The assignment cost c_{iq} for a_i is the number of shunting movements required if tram a_i is assigned to position p_q. The assignment cost c_{iq} can be determined as follows:

$$c_{iq} := \sum_{k=1}^{i-1} \sum_{p_l \in \mathcal{P}_r : l > q} x_{kl} \; + \sum_{k=i+1}^{N} \sum_{p_l \in \mathcal{P}_r : l < q} x_{kl}, \quad \text{where } p_q \in \mathcal{P}_r,\ a_i \in \mathcal{A}$$

❑

This sum represents the number of times that tram a_i has to be shunted with trams which are standing on-top of p_q but have arrived earlier or with trams which are standing below p_q but have arrived later.

If an assignment requires shunting, this can be illustrated by the following tripartite graph model. In this model, the vertex set V is defined as the union of $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{P}$. The arc set of this tripartite graph is determined by assignment matrices X and Y. A vertex a_i is connected to a position vertex p_q by an arc (a_i, p_q) if $x_{iq} = 1$, i.e., if the tram a_i is assigned to the stack position p_q. Analogously, a position vertex p_q is connected to a departure vertex d_j if $y_{jq} = 1$, i.e., if the tram standing at the stack position p_q is assigned to the departure d_j.

If we assume that the vertices of the three sets $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{D}$ are arranged in the same way as in Figure 4.1.2 and Figure 4.1.3, we can explain the shunting movements as follows. For the assignment matrices X and Y, shunting trams is necessary if and only if the corresponding arcs cross each other and the trams are assigned to the same stack. Figure 4.1.3 shows an example where the chosen assignment requires that three trams have to be shunted on arrival and two trams have to be shunted at departure.

Analogously to Definition 4.1.1, we define the cost for the assignment π_Y of trams to departures.

Definition 4.1.2: The cost of assigning a tram at stack position p_q to a departure d_j are defined by:

$$\bar{c}_{jq} := \sum_{k=1}^{j-1} \sum_{p_l \in \mathcal{P}_r : l < q} y_{kl} \; + \sum_{k=j+1}^{M} \sum_{p_l \in \mathcal{P}_r : l > q} y_{kl} \quad \text{for all } p_q \in \mathcal{P}_r,\ d_j \in \mathcal{D}$$

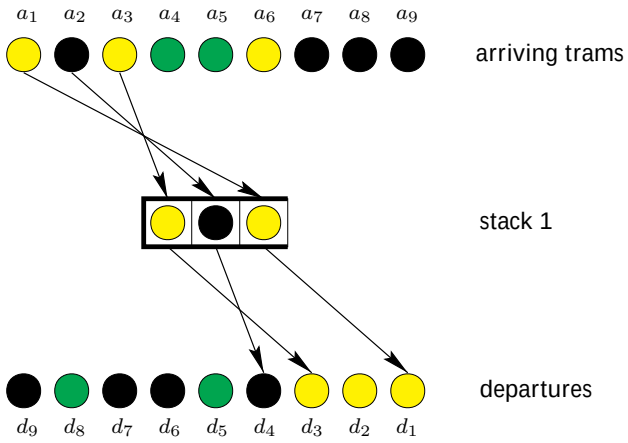

Figure 4.1.3: Shunting in a stack.

where y_{jq} is a binary variable and denotes that the tram standing at position p_q is assigned to departure d_j, $p_q \in \mathcal{P},\ d_j \in \mathcal{D}$.

$$y_{jq} = \begin{cases} 1 & \text{if the tram at position } p_q \text{ is assigned to departure } d_j \\ 0 & \text{otherwise} \end{cases} .$$

❑

We introduce the following two coefficients α_{ik} and β_{ql} for all indices $i, k, q, l \in \mathbb{N}$ where

$$\alpha_{ik} := \begin{cases} 1 & \text{if } i < k \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \beta_{ql} := \begin{cases} 1 & \text{if } q > l \text{ and } p_q, p_l \in \mathcal{P}_r \text{ for some } r \\ 0 & \text{otherwise} \end{cases}$$

Using this definition, the cost c_{iq} can be expressed by

$$c_{iq} = \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \big(\alpha_{ki}\beta_{lq} + \alpha_{ik}\beta_{ql}\big) x_{kl}.$$

We obtain the total of the assignment cost for the arrival part by

$$\sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} c_{iq} x_{iq} \quad = \quad \sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ki}\beta_{lq} x_{iq} x_{kl} + \sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql} x_{iq} x_{kl}$$

This sum is twice the number of shunting movements, because we have doubled the number of shunting movements by counting them twice, i.e., by counting them for both trams that have to be shunted. Exchanging the role of k and i as well as of q and l in the first sum yields

$$\sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} c_{iq} x_{iq} = 2 \cdot \sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql} x_{iq} x_{kl}$$

For the departure part, we apply the same argument to $\bar{c}_{jq}$ and obtain

$$\sum_{d_j \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} \bar{c}_{jq} y_{jq} = 2 \cdot \sum_{d_j \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} \sum_{d_k \in \mathcal{D}} \sum_{p_l \in \mathcal{P}} \alpha_{jk} \beta_{lq} y_{jq} y_{kl}$$

In both sums, we can neglect the factor of 2 because we have counted the shunting movements twice.

Consequently, we model the problem of minimizing the number of shunting movements, called **minimizing shunting problem MSP**, by the following binary quadratic program. In this problem, we take into account all the shunting movements that are necessary at arrival or at departure. The model turns out to be a combination of two quadratic assignment problems.

(MSP)

$$\min \quad \sum_{a_i \in \mathcal{A}} \sum_{a_k \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} \alpha_{ik} \beta_{ql}\, x_{iq}\, x_{kl} + \sum_{d_j \in \mathcal{D}} \sum_{d_k \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} \alpha_{jk} \beta_{lq}\, y_{jq}\, y_{kl} \tag{4.1.1}$$

$$\text{s.t.} \quad \sum_{a_i \in \mathcal{A}} x_{iq} = 1 \qquad \text{for all } \; p_q \in \mathcal{P} \tag{4.1.2}$$

$$\sum_{p_q \in \mathcal{P}} x_{iq} = 1 \qquad \text{for all } \; a_i \in \mathcal{A} \tag{4.1.3}$$

$$\sum_{d_j \in \mathcal{D}} y_{jq} = 1 \qquad \text{for all } \; p_q \in \mathcal{P} \tag{4.1.4}$$

$$\sum_{p_q \in \mathcal{P}} y_{jq} = 1 \qquad \text{for all } \; d_j \in \mathcal{D} \tag{4.1.5}$$

$$x_{iq} + y_{jq} \leq 1 \qquad \text{for all } \; p_q \in \mathcal{P},\; a_i \in \mathcal{A},\; d_j \in \mathcal{D} :\; t(a_i) \neq t(d_j) \tag{4.1.6}$$

$$x_{iq}, y_{jq} \in \{0,1\} \qquad \text{for all } \; a_i \in \mathcal{A},\; d_j \in \mathcal{D},\; p_q \in \mathcal{P} \tag{4.1.7}$$

where

$$\alpha_{ik} := \begin{cases} 1 & \text{if } i < k \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \beta_{ql} := \begin{cases} 1 & \text{if } q > l \text{ and } p_q, p_l \in \mathcal{P}_r \text{ for some } r \\ 0 & \text{otherwise} \end{cases}$$

The constraints (4.1.2) – (4.1.5) imply that each arriving tram is assigned to a position, each round trip is served, and to each position an arriving tram as well as a departure are assigned.

The constraints (4.1.6) describe the type requirements. If a tram a_i of type $t(a_i)$ is assigned to a position p_q, then only a departure d_j of the same type can be assigned to the same position. Thus, for d_j with $t(d_j) \neq t(a_i)$, y_{jq} must be equal to zero.

For these type constraints, alternative formulations can be given. If an arriving tram of type $\tau \in \mathcal{T}$ is assigned to p_q, then p_q must be assigned to a departure of the same type. This can be described by the following set of equations.

$$\sum_{a_i \in \mathcal{A}:\, t(a_i)=\tau} x_{iq} - \sum_{d_j \in \mathcal{D}:\, t(d_j)=\tau} y_{jq} = 0 \quad \text{for all } \; p_q \in \mathcal{P}, \tau \in \mathcal{T} \tag{4.1.8}$$

Another possible formulation is:

$$x_{iq} - \sum_{d_j \in \mathcal{D}:\, t(d_j)=t(a_i)} y_{jq} \leq 0 \quad \text{for all } a_i \in \mathcal{A},\; p_q \in \mathcal{P} \tag{4.1.9}$$

In this second formulation, we consider the possible assignment of tram a_i to position p_q represented by the variable x_{iq}. If $x_{iq} = 1$, a departure of the same type as a_i must be assigned to p_q. For reasons of symmetry, it is possible to define a similar set of constraints by exchanging the role of x and y.

The quadratic model MSP defined by (4.1.1) – (4.1.7) has some properties which we will use in the following sections. If we drop the type constraints (4.1.6), the MSP decomposes into two independent quadratic assignment problems (QAPs), describing the arrival and the departure part of the TDP. The corresponding subproblems are called the **arrival tram dispatch problem** (ATDP) and the **departure tram dispatch problem** (DTDP). Due to the structure of the shunting cost, both subproblems are similar. If we invert the order of the departure sequence, the subproblems become almost identical, differing only in the particular sequence of arrivals or departures.

In Section 4.1.1, we examine the arrival subproblem. We consider two well-known linearizations for quadratic assignment problems. For a recent survey on quadratic assignment problems and solution methods, we refer to Burkard et al. [BÇPP98] and Çela [Çel98]. The linearizations considered in this thesis are due to Frieze and Yadegar [FY83] and to Kaufman and Broeckx [KB78]. For the latter linearization method, we prove that the corresponding LP relaxation yields an optimal solution leading only to the trivial lower bound of zero for the integer program.

4.1.1 Quadratic Model for the Arrival Part

In this section, we consider the arrival subproblem that can be formulated by the following integer program.

$$\textbf{(ATDP)} \quad \min \quad \sum_{a_i \in \mathcal{A}} \sum_{a_k \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\, x_{iq}\, x_{kl} \tag{4.1.10}$$

$$\text{s.t.} \quad \sum_{a_i \in \mathcal{A}} x_{iq} = 1 \qquad \text{for all } p_q \in \mathcal{P} \tag{4.1.11}$$

$$\sum_{p_q \in \mathcal{P}} x_{iq} = 1 \qquad \text{for all } a_i \in \mathcal{A} \tag{4.1.12}$$

$$x_{iq} \in \{0,1\} \qquad \text{for all } a_i \in \mathcal{A},\ p_q \in \mathcal{P} \tag{4.1.13}$$

This representation shows that the arrival subproblem can be modeled as a 0-1 quadratic assignment problem (QAP). If we allow each tram to be assigned to every position, then there is an optimal solution of ATDP with an objective value of zero. This solution is easily achieved by setting $x_{iq} = 1$ for $i = q$ and $x_{iq} = 0$ for $i \neq q$. If we restrict the assignment of each tram to certain positions, solving ATDP becomes more difficult. For instance, if we fix for each position the assignment of a departure having a certain type, we must assign to each position a tram of the same type. The resulting problem has the same complexity as DTDP, i.e., it is $\mathcal{NP}$-hard. Both problems are symmetric in the way that, ATDP and DTDP differ only in the definition of shunting movements.

Since we are interested in solving TDP, we assume that we are given implicitly an assignment of departures to positions so that to each position only trams of specified type may be assigned. In the following, we consider linearization methods for the QAP which are used in a linearized model for TDP where the arrival part corresponding to ATDP is linearized. In Chapter 5, we make use of these linearization methods for DTDP, too.

There are several linearization methods for the QAP which might be useful in our case. We consider two linearizations. The first linearization is due to Frieze and Yadegar, the second was developed by Kaufman and Broeckx.

Linearization of Frieze and Yadegar

First, we consider the linearization of Frieze and Yadegar [FY83]. Frieze and Yadegar replace each product $x_{iq}\, x_{kl}$ by a binary variable z_{iqkl}. Applying the substitution to our quadratic model, we achieve the following linear binary program.

$$(\textbf{LATDP1})\ \min \sum_{a_i\in\mathcal{A}}\sum_{p_q\in\mathcal{P}}\sum_{a_k\in\mathcal{A}}\sum_{p_l\in\mathcal{P}} \alpha_{ik}\beta_{ql}\ z_{iqkl} \tag{4.1.14}$$

$$\text{s.t.} \sum_{a_i\in\mathcal{A}} z_{iqkl} = x_{kl} \qquad \text{for all } a_k \in \mathcal{A}, p_q, p_l \in \mathcal{P} \tag{4.1.15}$$

$$\sum_{p_q\in\mathcal{P}} z_{iqkl} = x_{kl} \qquad \text{for all } a_i, a_k \in \mathcal{A}, p_l \in \mathcal{P} \tag{4.1.16}$$

$$\sum_{a_k\in\mathcal{A}} z_{iqkl} = x_{iq} \qquad \text{for all } a_i \in \mathcal{A}, p_q, p_l \in \mathcal{P} \tag{4.1.17}$$

$$\sum_{p_l\in\mathcal{P}} z_{iqkl} = x_{iq} \qquad \text{for all } a_i, a_k \in \mathcal{A}, p_q \in \mathcal{P} \tag{4.1.18}$$

$$z_{iqiq} = x_{iq} \qquad \text{for all } a_i \in \mathcal{A}, p_q \in \mathcal{P} \tag{4.1.19}$$

$$0 \le z_{iqkl} \le 1 \qquad \text{for all } a_i, a_k \in \mathcal{A},\ p_q, p_l \in \mathcal{P} \tag{4.1.20}$$

We will come back to this linearization method when we consider computational results for MSP (cf. Section 4.4.1).

Kaufman and Broeckx linearization

Secondly, we linearize the quadratic assignment model for the arriving part using the approach of Kaufman and Broeckx [KB78]. Kaufman and Broeckx accumulate the cost of the assignment of an arriving tram a_i to stack position p_q by summing up all possible assignments of trams to the stack which would be in conflict with this tram.

Definition 4.1.3: For each $a_i \in \mathcal{A}$ and each $p_q \in \mathcal{P}$, we introduce a variable w_{iq} defined by

$$w_{iq} := x_{iq} \sum_{a_k\in\mathcal{A}}\sum_{p_l\in\mathcal{P}} \alpha_{ik}\beta_{ql}x_{kl}. \tag{4.1.21}$$

Additionally, for each $a_i \in \mathcal{A}$ and each $p_q \in \mathcal{P}$, we define a constant d_{iq} by

$$d_{iq} := \sum_{a_k\in\mathcal{A}}\sum_{p_l\in\mathcal{P}} \alpha_{ik}\beta_{ql}. \tag{4.1.22}$$

❑

Theorem 4.1.4: The quadratic integer program ATDP given by (4.1.10) – (4.1.13) is equivalent to the following linear mixed integer program:

(LATDP2)

$$\min \sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} w_{iq} \tag{4.1.23}$$

$$\text{s.t.} \sum_{a_i \in \mathcal{A}} x_{iq} = 1 \qquad \text{for all } p_q \in \mathcal{P} \tag{4.1.24}$$

$$\sum_{p_q \in \mathcal{P}} x_{iq} = 1 \qquad \text{for all } a_i \in \mathcal{A} \tag{4.1.25}$$

$$d_{iq}\, x_{iq} - w_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\, x_{kl} \le d_{iq} \quad \text{for all } a_i \in \mathcal{A},\ p_q \in \mathcal{P} \tag{4.1.26}$$

$$x_{iq} \in \{0,1\}, w_{iq} \ge 0 \qquad \text{for all } a_i \in \mathcal{A},\ p_q \in \mathcal{P} \tag{4.1.27}$$

Proof: The objective function (4.1.10) can easily be transformed to the objective function (4.1.23) using the Definition (4.1.21).

Let x^* be an optimal solution of ATDP. We define w^*_{iq} in accordance with (4.1.21), replacing x by x^*. Then, (x^*, w^*) is a feasible solution of LATDP2 (4.1.23) – (4.1.27), because

$$\begin{aligned} d_{iq}x^*_{iq} - w^*_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}x^*_{kl} &= d_{iq}x^*_{iq} - x^*_{iq} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}x^*_{kl} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}x^*_{kl} \\ &= d_{iq}x^*_{iq} + (1 - x^*_{iq}) \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}x^*_{kl} \\ &\le d_{iq}x^*_{iq} + (1 - x^*_{iq})\, d_{iq} \\ &= d_{iq}\ . \end{aligned}$$

It remains to show that the objective values of (4.1.10) with $x = x^*$ and (4.1.23) with $w = w^*$ are identical. Since w^* is defined in accordance with Definition (4.1.21), both objective values are equal.

Next, we assume that $(\hat{x}, \hat{w})$ is an optimal solution of LATDP2. Obviously, $\hat{x}$ is feasible for ATDP. We show that $\hat{x}$ is an optimal solution of ATDP.

In the first case, we assume that $\hat{w}_{iq} > 0$ and consider the constraint (4.1.26) which implies

$$d_{iq}\hat{x}_{iq} - d_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} \le \hat{w}_{iq}.$$

Since $\hat{w}_{iq} > 0$ and $\hat{w}_{iq}$ is minimal, it follows that

$$d_{iq}\hat{x}_{iq} - d_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} = \hat{w}_{iq}.$$

By definition of d_{iq}, we know that $d_{iq} \geq \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl}$. Consequently, $\hat{x}_{iq}$ must be equal to one and there must be an arriving tram $a_k \in \mathcal{A}$ and a position $p_l \in \mathcal{P}$ so that $\alpha_{ik}\beta_{ql} = \hat{x}_{kl} = 1$ in order to force $\hat{w}_{iq}$ to be positive. The optimality of $\hat{w}$ guarantees that

$$\begin{aligned}
\hat{w}_{iq} &= d_{iq}\hat{x}_{iq} - d_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} \\
&= \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} \\
&= \hat{x}_{iq} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl}.
\end{aligned}$$

Therefore, if $\hat{w}_{iq} > 0$, then $\hat{w}_{iq} = \hat{x}_{iq} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl}$.

In the second case, let $\hat{w}_{iq}$ be equal to zero. Let us assume that $\hat{x}_{iq} = 1$ and $\sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} \geq 1$. Then, the constraint (4.1.26)

$$d_{iq}\hat{x}_{iq} - d_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl} \leq 0$$

is violated. Hence, either $\hat{x}_{iq} = 0$ or, for all arrivals $a_k \in \mathcal{A}$ and positions $p_l \in \mathcal{P}$, the equality $\hat{x}_{kl} = 1$ implies that $\alpha_{ik}\beta_{ql} = 0$. Consequently, if $\hat{w}_{iq}$ is equal to zero, then $\hat{x}_{iq} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl}$ is equal to zero, too. Additionally, $\hat{w}_{iq} = \hat{x}_{iq} \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\hat{x}_{kl}$.

Consequently, the corresponding objective values for ATDP and LATDP2 are identical.

In both cases, an optimal solution of the first program implies a feasible solution of the other one. Furthermore, both solutions yield the same objective value. This completes the proof. ❑

4.1.2 Quadratic Model for the Departure Part

Analogously to the arrival part of the TDP, we derive a quadratic assignment model for the departure part. The departure problem can be described as follows: Given a fixed assignment of trams to the positions inside the depot, the objective is to find an assignment of stored trams to the round trips which minimizes the number of shunting movements necessary for the trams' departures.

The type of a tram stored at position p_q is predetermined by the assignment π_X of arriving trams to positions given by the assignment matrix $X = (x_{iq})$, i.e., $t(p_q) := t(a_i)$ if and only if $x_{iq} = 1$. By this definition, the function t is "extended" to $\mathcal{P}$.

$$\textbf{(DP)} \quad \min \quad \sum_{d_j \in \mathcal{D}} \sum_{d_k \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} \alpha_{jk} \beta_{lq} \, y_{jq} \, y_{kl} \tag{4.1.28}$$

$$\text{s.t.} \quad \sum_{d_j \in \mathcal{D}} y_{jq} = 1 \qquad \text{for all } p_q \in \mathcal{P} \tag{4.1.29}$$

$$\sum_{p_q \in \mathcal{P}} y_{jq} = 1 \qquad \text{for all } d_j \in \mathcal{D} \tag{4.1.30}$$

$$y_{jq} \in \{0,1\} \qquad \text{for all } d_j \in \mathcal{D},\ p_q \in \mathcal{P} \text{ and } t(d_j) = t(p_q) \tag{4.1.31}$$

An optimal solution of (DTDP) satisfies the following inequality for all stacks $\mathcal{P}_r$ and for all types $\tau \in \mathcal{T}$:

$$\begin{aligned} &\text{for all } d_j, d_k \in \mathcal{D} : j < k \quad \text{for all } p_q, p_l \in \mathcal{P}_r : q < l \\ &\text{where } t(d_j) = t(d_k) = t(p_q) = t(p_l) = \tau : \qquad y_{jq} + y_{kl} \leq 1. \end{aligned} \tag{4.1.32}$$

This inequality can be interpreted as follows: An optimal solution satisfies the property that if two trams of the same type are stored in the same stack, the top-most tram of both is assigned to the earlier departure (cf. Section 4.4.1).

4.1.3 Shunting of Trams

In this section, we examine more intensively the situation where shunting of trams is unavoidable. We prove that there is an optimal solution for TDP in which shunting only is necessary either when assigning the arriving trams to standing positions or during the departure of the trams.

We prove this result by giving a method to construct an assignment that requires shunting either on arrival or at departure. Based on an optimal solution, we construct such an assignment having the same cost.

In the following, we assume that π_X and π_Y are feasible assignments represented by the assignment matrices $X = (x_{iq})$ and $Y = (y_{jq})$.

We start with a lemma about unavoidable shunting movements.

Lemma 4.1.5: Let $x_{iq} = 1$ and $x_{kl} = 1$ for $i < k$ and $q > l + 1$ and $p_q, p_l \in \mathcal{P}_r$ for some stack r. Since $\alpha_{ik} = 1$ and $\beta_{ql} = 1$, a_k must be shunted with a_i to enter the stack in the order given by π_X.

Additionally, let a_j be the tram which is assigned to position p_{l+1}. Then, a_j has to be shunted with a_i or a_k or both.

Proof: We refer also to Figure 4.1.4.
1.) If $j < i$, then a_j and a_k must be shunted because $j < k$ and $l + 1 > l$.

2.) If $j > k$, then shunting of a_i and a_j is unavoidable since $i < j$ and $q > l + 1$.
3.) If $i < j < k$, then all three trams a_i, a_j, and a_k must be shunted pairwise. ❏

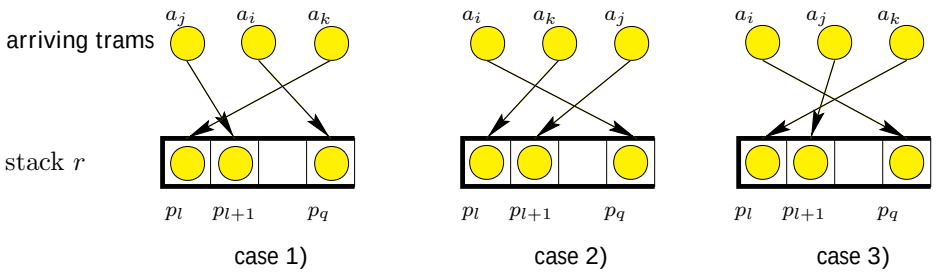

Figure 4.1.4: The three cases of Lemma 4.1.5

Corollary 4.1.6: Let (x, y) be a feasible solution of the TDP. If $y_{iq} = 1$ and $y_{kl} = 1$ for $i < k$ and $q < l - 1$ and $p_q, p_l \in \mathcal{P}_r$ for some r, then the trams at position p_q and p_l have to be shunted at departure.

Additionally, let the tram at position p_{l-1} be assigned to departure d_j. Then, the trams assigned to d_i and d_j or those assigned to d_j and d_k have to be shunted, too.

Assuming that shunting cannot be avoided, case 1 and 3 in the proof of Lemma 4.1.5 imply directly that two trams assigned to consecutive positions in a stack must be shunted. In the second case, the (position) distance between the two trams to be shunted is decreased by one. If we apply Lemma 4.1.5 iteratively to this situation, we obtain also a situation in which two trams at consecutive positions must be shunted.

Thus, Lemma 4.1.5 and Corollary 4.1.6 imply that if shunting cannot be avoided always two arriving trams assigned to consecutive positions in a stack or two trams standing at consecutive positions in a stack assigned to some departures must be shunted. Additionally, if the trams a_i and a_k are assigned to the positions p_q and p_l belonging to the same stack and have to be shunted, all the trams assigned to the positions p_v, $l < v < q$, also have to be shunted. The same holds for the departures d_i, d_j, and d_k assigned to the positions p_q, p_l, and p_v, $l > v > q$.

Consequently, shunting always occurs between two arrivals or departures that are assigned to two consecutive positions in a stack. We will make use of this fact when proving Theorem 4.1.7. We consider two positions p_q and p_{q+1} in the same stack $\mathcal{P}_r$ assuming that shunting is necessary for the departure of the two trams assigned to these positions. Given an optimal solution of the TDP, we construct a feasible assignment of the same cost where for these two positions the trams are shunted on arrival. Applying this transformation scheme, we can construct such an optimal solution of the TDP which proves the following theorem.

Theorem 4.1.7: There is an optimal solution of the MSP in which shunting is only necessary either for the arrival or the departure part.

Proof: We consider an optimal solution (x, y) of the MSP. If there is an optimal solution of the MSP where shunting is unnecessary, Theorem 4.1.7 holds. Therefore, we assume that shunting is unavoidable. By Lemma 4.1.5 and Corollary 4.1.6, there are two trams which have to be shunted and are either assigned to or standing at two consecutive positions in a stack.

For reasons of symmetry, we only need to show that if shunting is necessary during the departures, then it is possible to update the optimal assignment yielding the same objective value but avoiding shunting during the departures. We can apply a similar construction scheme to determine an assignment which avoids shunting on arrival.

Let a_i and a_j $(i < j)$ be such two trams assigned to positions p_q, p_{q+1} in some stack r and to departures $d_{i'}$ and $d_{j'}$, $i' < j'$.

We define the following sets of arriving trams, stack positions, and departures:

$A_1 := \{a_k \in \mathcal{A} \mid k < i\}$	trams arriving earlier than a_i
$A_2 := \{a_k \in \mathcal{A} \mid i < k < j\}$	trams arriving between a_i and a_j
$A_3 := \{a_k \in \mathcal{A} \mid j < k\}$	trams arriving later than a_j
$P_1 := \{p_k \in \mathcal{P}_r \mid k < q\}$	positions in stack r below p_q
$P_3 := \{p_k \in \mathcal{P}_r \mid q+1 < k\}$	positions in stack r on top of p_{q+1}
$D_1 := \{d_k \in \mathcal{D} \mid k < i'\}$	departures before $d_{i'}$
$D_2 := \{d_k \in \mathcal{D} \mid i' < k < j'\}$	departures between $d_{i'}$ and $d_{j'}$
$D_3 := \{d_k \in \mathcal{D} \mid j' < k\}$	departures after $d_{j'}$

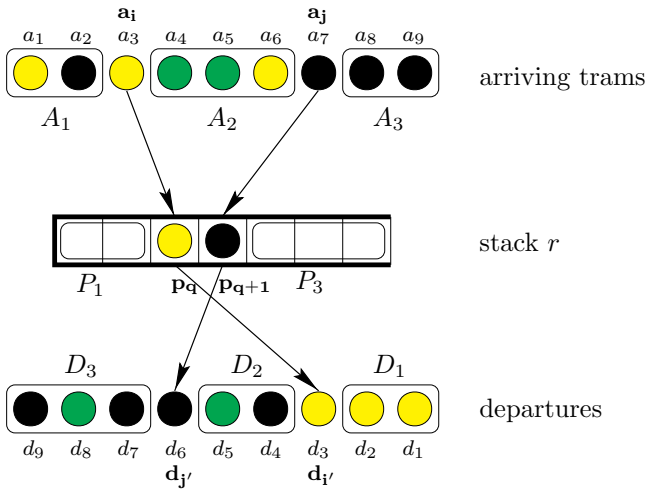

Figure 4.1.5: An example for the situation for tram a_i and a_j in Case 1.

By the following notations, we denote the number of trams in A_i assigned to

positions in P_q and the number of trams at positions in P_l assigned to departures in D_j :

$K^x_{iq} := \mid \{a_s \in A_i \mid x_{st} = 1 \text{ for } p_t \in P_q\} \mid$ for $i = 1, 2, 3$ and $q = 1, 3$

$K^y_{jl} := \mid \{d_s \in D_j \mid y_{st} = 1 \text{ for } p_t \in P_l\} \mid$ for $j = 1, 2, 3$ and $l = 1, 3$

In the following, we assume that at least two trams, a_i and a_j, have to be shunted when leaving the depot.

We assume that a_i and $d_{i'}$ are assigned to p_q whereas a_j and $d_{j'}$ are assigned to p_{q+1} . In terms of X and Y, we obtain that

$$\begin{aligned} x_{iq} &= 1 \\ y_{i'q} &= 1 \\ x_{j,q+1} &= 1 \\ y_{j',q+1} &= 1 \end{aligned}$$

Furthermore, we assume that both departures are of suitable type. Both trams can enter the stack in the order of their arrival, but have to be shunted before their departure. It is possible that other trams have to be shunted, too.

During the arrivals:

1. The trams of A_1 assigned to positions in P_3 have to be shunted with a_i and a_j.
2. The trams of A_2 assigned to positions in P_3 have to be shunted with a_j.
3. The trams of A_2 assigned to positions in P_1 have to be shunted with a_i.
4. The trams of A_3 assigned to positions in P_1 have to be shunted with tram a_i and tram a_j.

During the departures:

5. The trams standing at positions in P_1 and assigned to departures in D_1 have to be shunted with tram a_i at p_q and tram a_j at p_{q+1}.
6. The trams standing at positions in P_1 and assigned to departures in D_2 have to be shunted with tram a_j at position p_{q+1}.
7. The trams standing at positions in P_3 and assigned to departures in D_2 have to be shunted with tram a_i at p_q.
8. The trams standing at positions in P_3 and assigned to departures in D_3 have to be shunted with both trams a_i and a_j.
9. Tram a_i and tram a_j have to be shunted.

This means that if we restrict ourselves to this partial assignment for this stack, we need

$$2 \cdot K^x_{13} + K^x_{23} + K^x_{21} + 2 \cdot K^x_{31} + 2 \cdot K^y_{11} + K^y_{21} + K^y_{23} + 2 \cdot K^y_{33} + 1$$

shunting movements concerning tram a_i and/or tram a_j.

We define a new assignment represented by $(\hat{x}, \hat{y})$ in the way that we switch the positions to which a_i and a_j are assigned, i.e.,

$$\begin{aligned} \hat{x}_{i,q+1} &= 1 \qquad \hat{x}_{iq} = 0 \\ \hat{y}_{i',q+1} &= 1 \qquad \hat{y}_{i'q} = 0 \\ \hat{x}_{jq} &= 1 \qquad \hat{x}_{j,q+1} = 0 \\ \hat{y}_{j'q} &= 1 \qquad \hat{y}_{j',q+1} = 0 \end{aligned}$$

The remaining part of the solution is left unchanged which means that for all other indices, we define $\hat{x}$ and $\hat{y}$ identically to x and y, i.e.,

$$\begin{aligned} \hat{x}_{kl} = x_{kl} \quad &\text{for all} \quad k \neq i, j, \quad l \neq q, q+1 \quad \text{and} \\ \hat{y}_{kl} = y_{kl} \quad &\text{for all} \quad k \neq i', j', \quad l \neq q, q+1. \end{aligned}$$

The arriving trams are still assigned to the same departures but have now changed their stack positions.

If we consider the objective value of this new assignment, we observe that we need the same number of shunting movements for this new assignment as for the given optimal assignment.

If $K^y_{23} + K^y_{11} + K^y_{21} + K^y_{33} \geq 1$ at least one shunting movement during the departure is left for stack r concerning the tram at position p_q or p_{q+1} or both trams. Once again, by Corollary 4.1.6, there are two trams that have to be shunted and are standing at consecutive positions in stack r.

The new solution $(\hat{x}, \hat{y})$ requires the same number of shunting movements as x and y but one shunting less on the departure part. If there are still two trams a_k and a_l to be shunted at departure and standing at two consecutive positions in a stack, we apply again the same construction scheme.

By Lemma 4.1.5 and Corollary 4.1.6, there must be such two trams if shunting is necessary for the departure part unless all the departures are possible without shunting. We apply the construction scheme iteratively, until all shunting movements are transformed from the departure part to the arrival part. ❑

As a consequence of Theorem 4.1.7, we derive a simplified model for the MSP. We will make use of the second linearization method LATDP2 for ATDP (see Theorem 4.1.4). Additionally, we suppose that trams are shunted only immediately at their arrival. This is motivated by the short time intervals between two consecutive departures. Usually, shunting is impossible at departure because of the lack of time.

We use constraint (4.1.8) instead of (4.1.6) because of the better performance in the computational tests. The constraints (4.1.34) motivated by the feasible

inequalities (4.1.32) are also inserted in order to improve the computation time. The constraints (4.1.36) enforce that all departures are possible without shunting.

(LADP)

$$\min \sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}} w_{iq} \tag{4.1.33}$$

$$\text{s.t.} \sum_{a_i \in \mathcal{A}} x_{iq} = 1 \qquad \forall p_q \in \mathcal{P} \tag{4.1.2}$$

$$\sum_{p_q \in \mathcal{P}} x_{iq} = 1 \qquad \forall a_i \in \mathcal{A} \tag{4.1.3}$$

$$x_{iq} + x_{jl} \leq 1 \qquad \forall a_i, a_j \in \mathcal{A} : i < j \wedge t(a_i) = t(a_j), \; \forall r \; \forall p_q, p_l \in \mathcal{P}_r : \; q > l \tag{4.1.34}$$

$$\sum_{a_j \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ij} \beta_{ql} \, x_{jl} + d_{iq} \, x_{iq} - w_{iq} \leq d_{iq} \quad \forall a_i \in \mathcal{A}, \; p_q \in \mathcal{P} \tag{4.1.35}$$

$$\sum_{d_j \in \mathcal{D}} y_{jq} = 1 \qquad \forall p_q \in \mathcal{P} \tag{4.1.4}$$

$$\sum_{p_q \in \mathcal{P}} y_{jq} = 1 \qquad \forall d_j \in \mathcal{D} \tag{4.1.5}$$

$$y_{iq} + y_{jl} \leq 1 \qquad \forall d_i, d_j \in \mathcal{D} : \; i < j, \; \forall r \; \forall p_q, p_l \in \mathcal{P}_r : q < l \tag{4.1.36}$$

$$\sum_{\substack{a_i \in \mathcal{A}: \\ t(a_i) = \tau}} x_{iq} - \sum_{\substack{d_j \in \mathcal{D}: \\ t(d_j) = \tau}} y_{jq} = 0 \qquad \forall p_q \in \mathcal{P}, \; \tau \in \mathcal{T} \tag{4.1.8}$$

$$x_{iq} \in \{0, 1\} \qquad \forall a_i \in \mathcal{A}, \; p_q \in \mathcal{P} \tag{4.1.7.1}$$

$$y_{jq} \in \{0, 1\} \qquad \forall d_j \in \mathcal{D}, \; p_q \in \mathcal{P} \tag{4.1.7.2}$$

$$w_{iq} \geq 0 \qquad \forall a_i \in \mathcal{A}, \; p_q \in \mathcal{P} \tag{4.1.37}$$

Next, we consider the LP relaxation of LADP (4.1.2) – (4.1.7.2). We obtain the following Lemma.

Lemma 4.1.8: For each $a_i \in \mathcal{A}$ and each position $p_q \in \mathcal{P}$, we define $\bar{x}_{iq} = \frac{1}{P} = \frac{1}{|\mathcal{P}|}$ and $\bar{w}_{iq} = 0$. Additionally, for each $d_j \in \mathcal{D}$ and each position $p_q \in \mathcal{P}$,

we define $\bar{y}_{jq} = \frac{1}{P}$. Then, $(\bar{x}, \bar{y}, \bar{w})$ is an optimal solution of the LP relaxation of LADP defined by (4.1.2) – (4.1.7.2).

Proof: Since $\mid \mathcal{D} \mid = \mid \mathcal{A} \mid = \mid \mathcal{P} \mid = P$, it is obvious that the constraints (4.1.2), (4.1.3), (4.1.4), and (4.1.5) are satisfied for all $a_i \in \mathcal{A}$, $p_q \in \mathcal{P}$, and $d_j \in \mathcal{D}$.

For all types $\tau \in \mathcal{T}$ there are as many trams of type τ as departures of the same type. Consequently, the type constraints (4.1.8) hold because $x_{iq} = y_{jq}$ for all $a_i \in \mathcal{A}$, $p_q \in \mathcal{P}$, and $d_j \in \mathcal{D}$.

Since

$$d_{iq}\bar{x}_{iq} - \bar{w}_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\bar{x}_{kl} \overset{(4.1.22)}{=} \frac{2}{P} d_{iq}, \qquad (*)$$

the constraint (4.1.35)

$$d_{iq}\bar{x}_{iq} - \bar{w}_{iq} + \sum_{a_k \in \mathcal{A}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{ql}\bar{x}_{kl} \leq d_{iq}$$

is satisfied for $P \geq 2$ and for all $a_i \in \mathcal{A}$ and $p_q \in \mathcal{P}$.

For a single stack of length one, no shunting movement is required. If $P = 1$ then $q = l = 1$ and $\beta_{ql} = 0$ implies that $d_{iq} = 0$.

Therefore, $(*)$ implies that $\bar{x}$ is feasible for the LP relaxation of LADP.

By the same argument, $\bar{y}$ satisfies the constraints (4.1.36) for $P = 1$. If $P \geq 2$, (4.1.36) is also satisfied because $y_{jq} \leq 2$ for all $d_j \in \mathcal{D}$ and all $p_q \in \mathcal{P}$.

Since $\bar{w}_{iq} = 0$ for all $a_i \in \mathcal{A}$ and $p_q \in \mathcal{P}$, $(\bar{x}, \bar{y}, \bar{w})$ is an optimal solution for the LP relaxation of LADP. ❑

Remark 4.1.9: The solution $(\bar{x}, \bar{y}, \bar{w})$ yields an objective value of 0. This means that, by the LP relaxation of LADP, we can only obtain the trivial lower bound.

4.1.4 Modifications

In the last section, we assumed that all trams arriving at the depot have to leave the depot in the next schedule period. Additionally, we assumed that there are exactly as many depot position as trams are to be stored.

Now, we generalize the situation and consider different modifications. First, we consider the case that there are more positions than trams. Secondly, we focus on situations where some trams were not used in the previous schedule period and some trams will not be used in the next schedule period. The trams of the first group are stored in the depot before the first tram arrives at the depot after serving the round trip. The trams of the second group stay in the depot after the last departure. In particular, such a situation occurs at the weekends where less trams are needed for the weekend schedule(s). Thirdly, we consider the case where the trams have different lengths.

More Positions than Trams

We start with the case that there are more positions in the depot than needed to store the trams. We assume that the depot is empty before the first arrival and after the last departure. We also assume that $\mid \mathcal{D} \mid \leq \mid \mathcal{A} \mid \leq \mid \mathcal{P} \mid$.

Not every stack position is used for storing the trams. Hence, the constraints (4.1.2) and (4.1.4) have to be modified to

$$\sum_{a_i \in \mathcal{A}} x_{iq} \leq 1 \qquad \text{for all } \; p_q \in \mathcal{P} \quad (4.1.2')$$

$$\sum_{d_j \in \mathcal{D}} y_{jq} \leq 1 \qquad \text{for all } \; p_q \in \mathcal{P} \quad (4.1.4')$$

Since we model the assignment of a tram to a stack position and to a departure by the two assignments $\pi_X : \mathcal{A} \to \mathcal{P}$ and $\pi_Y : \mathcal{D} \to \mathcal{P}$, we have to guarantee that $\pi_X(\mathcal{A}) \supseteq \pi_Y(\mathcal{D})$. This means that a departure is assigned to a position only if an arriving tram is assigned to this position. Consequently,

$$\sum_{a_i \in \mathcal{A}} x_{iq} \geq \sum_{d_j \in \mathcal{D}} y_{jq} \qquad \text{for all } \; p_q \in \mathcal{P} \tag{4.1.38}$$

Note that (4.1.2') and (4.1.38) imply (4.1.4').

Trams Staying in the Depot

In the next step, we assume that we do not need all the trams to serve the round trips of the next schedule. $\mid \mathcal{A} \mid - \mid \mathcal{D} \mid > \; 0$ trams stay in the depot after the last departure. Once again, we assume that the depot is empty before the first arrival. This situation is already modeled by (4.1.2') and (4.1.38). If $\mid \mathcal{A} \mid = \mid \mathcal{D} \mid$, $\sum_{a_i \in \mathcal{A}} x_{iq}$ equals $\sum_{d_j \in \mathcal{D}} y_{jq}$ for all positions p_q. If $\mid \mathcal{A} \mid > \mid \mathcal{D} \mid$, then (4.1.38) allows that some trams stored in the stacks stay in the depot. If some trams have stayed in the depot, we are faced with the situation that the depot is not empty before the trams having served the actual schedule period arrive at the depot.

In the following, we assume that the local transport company operates N trams. The set $\mathcal{A}$ consists of N trams and $\mathcal{D}$ consists of N departures. $\mathcal{A}$ is partitioned into two subsets $\mathcal{A}_1 = \{a_1, \ldots, a_{N_1}\}$ and $\mathcal{A}_2 = \{a_{N_1+1}, \ldots, a_N\}$ where $\mathcal{A}_1$ contains the $N_1 \leq N$ trams staying in the depot before the arrival of the $N_2 = N - N_1$ trams of $\mathcal{A}_2$. $\mathcal{D}_1 = \{d_1, \ldots, d_{M_1}\}$ consists of the M_1 departures of the next schedule period. The set $\mathcal{D}_2 = \{d_{M_1+1}, \ldots, d_N\}$ is a set of $M_2 = N - M_1$ "dummy" departures. The trams assigned to departures of $\mathcal{D}_2$ stay in the depot. Note that in an iterative process from day to day, the trams assigned to $\mathcal{D}_2$ form the set $\mathcal{A}_1$ of the next day.

We are now able to model this situation as a shunting problem where $\mid \mathcal{A} \mid = \mid \mathcal{D} \mid \leq \mid \mathcal{P} \mid$ (cf. Figure 4.1.6).

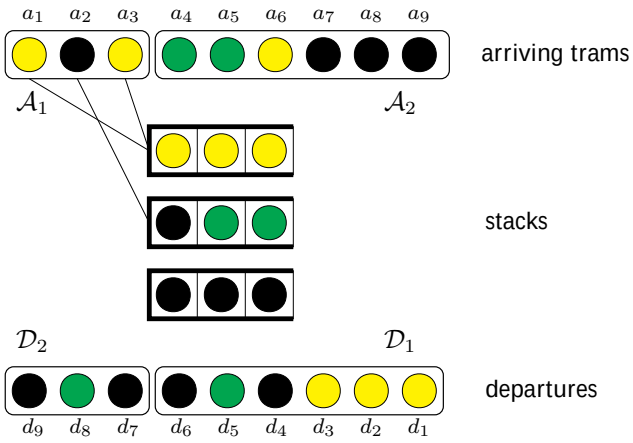

Figure 4.1.6: Trams standing in the depot.

The trams of $\mathcal{A}_1$ stay already in the depot. Hence, there are no shunting movements required between two trams of $\mathcal{A}_1$. The same holds for the trams of $\mathcal{D}_2$ which do not leave the depot. In the actual schedule period, shunting between these trams is not necessary at departure.

Hence, the coefficients α_{ik} have to be adapted to this situation. For instance, for a_i and a_k in $\mathcal{A}_1$, $\alpha_{ik}^{\mathcal{A}} := 0$. For d_j and d_k in $\mathcal{D}_2$, $\alpha_{jk}^{\mathcal{D}} := 0$. This is done separately for the arrival and the departure part because M_1 does not necessarily have to be equal to M_2 and the indices of both sets do not match.

Trams of Different Lengths

In general, local transport companies operate trams of different type and of different lengths. Since the capacity of each stack in which the trams are stored is bounded, we may have to take care of the tram lengths when assigning the trams to the stack positions.

We denote by $l_i \in \mathbb{N}$ the length of each tram $a_i \in \mathcal{A}$, $1 \leq i \leq N$. The length of stack r is denoted by $L_r \in \mathbb{N}$, $1 \leq r \leq R$.

Then the trams assigned to stack r have to satisfy the following knapsack constraint

$$\sum_{a_i \in \mathcal{A}} \sum_{p_q \in \mathcal{P}_r} l_i \, x_{iq} \leq L_r \quad \text{for all stacks } r \tag{4.1.39}$$

Using the modifications introduced in this section, MSP can be updated and applied to these situations.

4.2 Minimizing Type Mismatches

In this section, we investigate a variation of the TDP: the type mismatch problem TMP (cf. Definition 3.3.15). Instead of seeking for a solution which minimizes the amount of shunting, we are looking for an assignment of arriving trams to depot positions and to departures which does not require any shunting of trams. Obviously, such an assignment that satisfies also the type constraints does not necessarily exist. If we insist on avoiding shunting movements, we have to relax our type requirements. A possible optimization approach is the following. We seek for a solution that avoids shunting and minimizes the number of departures to which a tram of unsuitable type is assigned. We call this problem **(minimizing) type mismatch problem** (TMP). An instance of TMP consists of

- a set of incoming trams $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$,
- a set of round trips (departures) $\mathcal{D} = \{d_1, d_2, \dots, d_M\}$ that have to be served,
- a set of (stack) positions $\mathcal{P} = \{p_1, p_2, \dots, p_P\}$ to which the arriving trams have to be assigned, and
- R stacks in which the positions are located.

Analogously to the introduction of an instance of TDP (cf. Section 4.1), each tram and each departure has a type $\tau \in \mathcal{T} = \{\tau_1, \dots, \tau_T\}$. The type of a tram and a departure is specified by the function $t : \mathcal{A} \cup \mathcal{D} \to \mathcal{T}$. The positions in $\mathcal{P}$ are arranged in R stacks in the same way as introduced in Section 4.1.

Using the notations of the previous section, we can define a **type mismatch** as follows. Assume that to a position $p_q \in \mathcal{P}$ an arriving tram $a_i \in \mathcal{A}$ of type $t(a_i)$ is assigned. A **type mismatch** occurs if this trams standing at position p_q is assigned to a departure $d_j \in \mathcal{D}$ of type $t(d_j) \neq t(a_i)$. A possible objective function counts the total number of positions where $t(a_i) \neq t(d_j)$ for $a_i \in \mathcal{A}, d_j \in \mathcal{D}$, and $x_{iq} = 1$, $y_{jq} = 1$.

By θ_{ij} and $\bar{\theta}_{ij}$, we indicate whether the types of $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ match. For all $a_i \in \mathcal{A}$ and all $d_j \in \mathcal{D}$, the coefficients θ_{ij} indicate whether a_i and d_j have the same type:

$$\theta_{ij} := \begin{cases} 1 & \text{if } t(a_i) = t(d_j), a_i \in \mathcal{A}, d_j \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases} .$$

For all $a_i \in \mathcal{A}$ and all $d_j \in \mathcal{D}$, we define $\bar{\theta}_{ij} := 1 - \theta_{ij}$ to indicate that the types of a_i and d_j do not match.

For $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ with $\bar{\theta}_{ij} = 1$, a type mismatch occurs if $x_{iq} = y_{jq} = 1$ for some position $p_q \in \mathcal{P}$. Hence, a possible objective is to maximize the number

of positions for which x_{iq} and y_{jq} differ. This leads to

$$\begin{aligned}\sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\bar{\theta}_{iq}\mid x_{iq}-y_{jq}\mid &= \sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}(1-\theta_{ij})\mid x_{iq}-y_{jq}\mid\\ &= \sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\mid x_{iq}-y_{jq}\mid\\ &\quad -\sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\theta_{ij}\mid x_{iq}-y_{jq}\mid\end{aligned}$$

The first term $\sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\mid x_{iq}-y_{jq}\mid$ is a constant because for each position there is exactly one variable x_{iq} and one variable y_{jq} which is equal to one. The concrete value of this term is $2N(N-1)$ where N denotes the number of trams which is assumed to be equal to the number of positions and departures. Consequently, maximizing

$$\sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\bar{\theta}_{iq}\mid x_{iq}-y_{jq}\mid$$

is equivalent to minimizing

$$\sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\theta_{ij}\mid x_{iq}-y_{jq}\mid .$$

We can model the type mismatch problem by the following binary linear program:

(MTMP)

$$\min \sum_{p_q\in\mathcal{P}}\sum_{a_i\in\mathcal{A}}\sum_{d_j\in\mathcal{D}}\theta_{ij}\mid x_{iq}-y_{jq}\mid \tag{4.2.40}$$

$$\text{s.t.}\sum_{a_i\in\mathcal{A}}x_{iq}=1 \qquad \text{for all } p_q\in\mathcal{P} \tag{4.2.41}$$

$$\sum_{p_q\in\mathcal{P}}x_{iq}=1 \qquad \text{for all } a_i\in\mathcal{A} \tag{4.2.42}$$

$$x_{iq}+x_{jl}\le 1 \qquad \text{for all } a_i,a_j\in\mathcal{A}: i<j,\ r,\ p_q,p_l\in\mathcal{P}_r: q>l \tag{4.2.43}$$

$$\sum_{p_q\in\mathcal{P}}y_{jq}=1 \qquad \text{for all } d_j\in\mathcal{D} \tag{4.2.44}$$

$$\sum_{d_j\in\mathcal{D}}y_{jq}=1 \qquad \text{for all } p_q\in\mathcal{P} \tag{4.2.45}$$

$$y_{iq}+y_{jl}\le 1 \qquad \text{for all } d_i,d_j\in\mathcal{D}: i<j,\ r,\ p_q,p_l\in\mathcal{P}_r: q<l \tag{4.2.46}$$

$$x_{iq}\in\{0,1\}\quad y_{jq}\in\{0,1\} \qquad \text{for all } a_i\in\mathcal{A}, d_j\in\mathcal{D}, p_q\in\mathcal{P} \tag{4.2.47}$$

The objective function (4.2.40) counts for each position $p_q \in \mathcal{P}$ the number of times that x_{iq} differs from y_{jq} for $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ with $t(a_i) = t(d_j)$. If for a particular position p_q both values x_{iq} and y_{jq} differ for all i and all j except for those for which both values are zero, then there is a type mismatch at position p_q. The objective value of (4.2.40) is not equal to the number of type mismatches because it counts each type mismatch twice and it contains an additive constant.

Theorem 4.2.10: An optimal solution of the binary program MTMP (4.2.40)-(4.2.47) corresponds to an optimal solution of the problem to minimize type mismatch. A feasible solution of the MTMP avoids shunting completely. A lower bound for the objective function of the MTMP is given by

$$2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau \cdot \big(K_\tau - 1\big)$$

where $K_\tau :=\mid \{a \in \mathcal{A} \mid t(a) = \tau\} \mid$. An upper bound is given by $2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau^2$.

Remark 4.2.11: In this section, we force the number of arrivals N to be equal to the number of departures M and to the number of stack positions P. Furthermore, we assume that for all $\tau \in \mathcal{T}$ the number of arriving trams of type τ equals the number of departures $d \in \mathcal{D}$ of the same type. Otherwise, there is no assignment that satisfies the type constraints, i.e., it is impossible to serve all the scheduled round trips with trams of specified type.

By this assumption $\{a \in \mathcal{A} \mid t(a) = \tau\}$ is of the same cardinality as $\{d \in \mathcal{D} \mid t(d) = \tau\}$. Consequently, $K_{t(a_i)} = \mid \{a \in \mathcal{A} \mid t(a) = t(a_i)\} \mid = \mid \{d_j \in \mathcal{D} \mid t(d_j) = t(a_i)\} \mid = \sum_{d_j \in \mathcal{D}} \theta_{ij}$ for all $a_i \in \mathcal{A}$ and $K_{t(d_j)} = \sum_{a_i \in \mathcal{A}} \theta_{ij}$ for all $d_j \in \mathcal{D}$.

θ_{ij} indicates if the types of $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ are equal whereas $\bar{\theta}_{ij} = 1$ means that both types are different.

We call two assignments π_X and π_Y, represented by the correspondent assignment matrices $X = (x_{iq})$ and $Y = (y_{jq})$, **feasible** for MTMP if they satisfy (4.2.41) – (4.2.47).

If π_X and π_Y are feasible, then for each position p_q there are two indices $i(q)$ and $j(q)$ for which $x_{i(q)q} = 1$ and $y_{j(q)q} = 1$. A type mismatch at position p_q implies that $\bar{\theta}_{i(q)j(q)} = 1$. If $\bar{\theta}_{i(q)j(q)} = 1$, then the types of $a_{i(q)}$ and $d_{j(q)}$ are different implying a type mismatch at position p_q.

Proof of Theorem 4.2.10: We start the proof by showing that a feasible solution of the MTMP avoids shunting completely. Then, we continue by deriving a lower bound on the objective function (4.2.40) and proving implicitly that the binary program is correct.

In the following, we assume that π_X and π_Y are feasible for MTMP.

Shunting: Shunting of arriving trams is necessary if two trams a_i and a_k, $i < k$, are assigned to the same stack and a_i is assigned to an on-top position of a_k.

Furthermore, shunting cannot be avoided if two departures d_j and d_l, $j < l$, are served by trams standing in the same stack where d_j is served by a tram assigned to a deeper position inside the stack than the tram assigned to d_l. It is easy to see that these conditions give a complete description of the situations in which shunting of trams is necessary. By the constraints (4.2.43) and (4.2.46), such assignments are forbidden. Consequently, a feasible solution of the MTMP avoids shunting completely.

Lower Bound: By (4.2.41) to (4.2.45) there must be a unique index $i(q)$ and a unique index $j(q)$ for each position p_q with $x_{i(q)q} = 1$ and $y_{j(q)q} = 1$ whereas $x_{iq} = 0$ and $y_{jq} = 0$ for all $a_i \in \mathcal{A}$ with $i \neq i(q)$ and all $d_j \in \mathcal{D}$ with $j \neq j(q)$.

The objective function (4.2.40) can be transformed as follows:

$$
\begin{aligned}
& \sum_{p_q \in \mathcal{P}} \sum_{a_i \in \mathcal{A}} \sum_{d_j \in \mathcal{D}} \theta_{ij} \mid x_{iq} - y_{jq} \mid \\
= \; & \sum_{p_q \in \mathcal{P}} \Big(\sum_{\substack{a_i \in \mathcal{A}: \\ i \neq i(q)}} \sum_{\substack{d_j \in \mathcal{D}: \\ j \neq j(q)}} \theta_{ij} \mid x_{iq} - y_{jq} \mid \; + \theta_{i(q)j(q)} \mid x_{i(q)q} - y_{j(q)q} \mid \\
& + \sum_{\substack{a_i \in \mathcal{A}: \\ i \neq i(q)}} \theta_{ij(q)} \mid x_{iq} - y_{j(q)q} \mid + \sum_{\substack{d_j \in \mathcal{D}: \\ j \neq j(q)}} \theta_{i(q)j} \mid x_{i(q)q} - y_{jq} \mid \Big) \\
= \; & \sum_{p_q \in \mathcal{P}} \Big(\sum_{\substack{a_i \in \mathcal{A}: \\ i \neq i(q)}} \theta_{ij(q)} + \sum_{\substack{d_j \in \mathcal{D}: \\ j \neq j(q)}} \theta_{i(q)j} \Big) \\
= \; & \sum_{p_q \in \mathcal{P}} \Big(\sum_{a_i \in \mathcal{A}} \theta_{ij(q)} - \theta_{i(q)j(q)} + \sum_{d_j \in \mathcal{D}} \theta_{i(q)j} - \theta_{i(q)j(q)} \Big) \\
= \; & \sum_{p_q \in \mathcal{P}} \Big(\sum_{a_i \in \mathcal{A}} \theta_{ij(q)} - 1 + \sum_{d_j \in \mathcal{D}} \theta_{i(q)j} - 1 \Big) + 2 \cdot \sum_{p_q \in \mathcal{P}} \bar{\theta}_{i(q)j(q)} \\
= \; & \sum_{p_q \in \mathcal{P}} \big(K_{t(d_{j(q)})} - 1 + K_{t(a_{i(q)})} - 1 \big) + 2 \cdot \sum_{p_q \in \mathcal{P}} \bar{\theta}_{i(q)j(q)} \\
= \; & 2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau \big(K_\tau - 1 \big) + 2 \cdot \sum_{p_q \in \mathcal{P}} \bar{\theta}_{i(q)j(q)}
\end{aligned}
$$

This yields the lower bound of $2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau (K_\tau - 1)$ for the objective function (4.2.40).

By the last equation, we observe that this lower bound is tight if there are no type mismatches at any position $p_q \in \mathcal{P}$. For each type mismatch at a position $p_q \in \mathcal{P}$, the objective function (4.2.40) increases by 2.

Upper Bound: A trivial upper bound for the objective function is given by

$$
2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau \cdot \big(K_\tau - 1 \big) + 2 \cdot P = 2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau \cdot \big(K_\tau - 1 \big) + 2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau = 2 \cdot \sum_{\tau \in \mathcal{T}} K_\tau^2.
$$

This completes the proof. ❑

We observe that the lower bound derived by solving the LP relaxation of the MTMP yields worse bounds than those given in Theorem 4.2.10:

Theorem 4.2.12: An optimal solution of the LP relaxation of MTMP has an objective value of zero.

Proof: We prove the theorem by giving a feasible solution of MTMP having objective value zero. Since the objective function of MTMP is non-negative, this solution is optimal.

Without loss of generality, we assume that $\mid \mathcal{P} \mid \geq 2$. For instances of one tram, one position, and one departure, our assumption that for all $\tau \in \mathcal{T}$ the number of trams of type τ is equal to the number departures of type τ implies that there is a shunting-free and type-preserving solution for TMP. In this case, the optimal objective value of MTMP (and its LP relaxation) is zero.

For all trams $a_i \in \mathcal{A}$, all departures $d_j \in \mathcal{D}$, and all positions $p_q \in \mathcal{P}$, we define $x_{iq} = \frac{1}{|\mathcal{P}|}$ and $y_{jq} = \frac{1}{|\mathcal{P}|}$.

The constraints (4.2.41), (4.2.42), (4.2.44), and (4.2.45) are satisfied, since for MTMP we assumed that $\mid \mathcal{A} \mid = \mid \mathcal{P} \mid = \mid \mathcal{D} \mid$. The constraints (4.2.43) and (4.2.46) are satisfied because $\mid \mathcal{P} \mid \geq 2$. Consequently, the considered solution is feasible for MTMP.

For every position $p_q \in \mathcal{P}$, the difference $x_{iq} - y_{iq}$ is equal to zero. Hence, the corresponding objective value is zero so that (x, y) is an optimal solution of the LP relaxation of MTMP. ❑

4.3 Relations Between Solutions for TDP and TMP

In the following, we assume that we are given feasible solutions for TMP and TDP. Given a solution for TDP (respectively TMP), we construct a feasible solution for TMP (respectively TDP) and examine the corresponding objective value. We achieve the following theorems:

Theorem 4.3.13: Let π_X and π_Y denote the assignments of a feasible solution with objective value s for TDP. Then, there is a feasible solution of TMP with an objective value of at most $2s$.

Proof: By Theorem 4.1.7, if suffices to consider an optimal solution for TDP which is shunting-free, for instance, at arrival. By feasibility, the assignments π_X and π_Y are type-preserving.

If two trams stored at two consecutive positions in a stack have to be shunted at departure, we can reduce the number of shunting movements by exchanging the assignment π_Y for both positions. The assignment π_X is left unchanged.

The resulting assignment π'_Y requires one shunting movement less and (at most) two type mismatches more than π_Y.

We repeat this switching procedure for all pairs of trams that require shunting at departure until π'_Y is shunting-free. For each shunting movement, we obtain at most two type mismatches resulting in a feasible solution for TMP of at most $2s$ type mismatches. ❑

Better upper bounds for the TMP solution may be calculated by using more sophisticated switching approaches when constructing a feasible solution from a feasible solution of TDP. Not every switch in the assignment π_Y may require two additional type mismatches. Next, we assume that we are given a feasible assignment for TMP.

Theorem 4.3.14: Let π_X and π_Y denote the assignments of a feasible solution with objective value t for TMP. Then, there is a feasible solution of TDP with an objective value of $t \cdot \max_r P_r$ where P_r denotes the length of stack $\mathcal{P}_r$, $1 \leq r \leq R$.

Proof: By feasibility for TMP, π_X and π_Y are shunting-free. Let d_j be a departure of type $\tau \in \mathcal{T}$ assigned to a position $p_q \in \mathcal{P}$ in some stack $\mathcal{P}_\nu$. Moreover, we assume that to the same position, a tram a_i of non-matching type is assigned. Since $\mid \{a_i \in \mathcal{A} \mid t(a_i) = \tau\} \mid = \mid \{d_j \in \mathcal{D} \mid t(d_j) = \tau\} \mid$, we can find a tram of the same type that is assigned to a type mismatching departure $d_k \in \mathcal{D}$ and to a position p_l in some stack $\mathcal{P}_r$ of length P_r.

We change π_Y in the following way: We assign d_j to p_l which results in at most P_r shunting movements and reduce the number of type mismatches by one. If the type of a_i matches to the type of d_k, then we assign d_k to p_q which results again in at most P_ν shunting movements and reduces the number of type mismatches by one. Otherwise, we leave d_k and p_q unassigned and consider both as belonging to the same type mismatch.

We repeat this procedure for every remaining type mismatch. The resulting solution is type-preserving and requires at most $t \cdot \max_r P_r$ type mismatches. ❑

When constructing a feasible solution for TDP, the number of shunting movement is in fact at most the sum of the lengths of the stacks that are involved in the update of π_Y. The length of such a stack appears in this sum as often as the stack is involved in the update. Moreover, the length P_r is only a rough estimate for the number of shunting movement required by the update of π_Y.

4.4 Algorithms and Computational Results

In this section, we present different exact and heuristic algorithms for TDP and TMP. The tram dispatch problems at departure DTDP and DTMP will be considered in the next chapter. We present computational results for real-world as well as for randomly generated data.

4.4.1 Exact Algorithms for TDP

In Section 4.1.1, we have introduced two different linearization methods for quadratic assignment problems as they occur in the context of the tram dispatching problem TDP. For the arrival subproblem (cf. Section 4.1.1), we introduced the linearization method of Frieze and Yadegar, denoted by LATDP1, and the linearization method of Kaufman and Broeckx, denoted by LADTP2. By Theorem 4.1.7, there is an optimal solution of TDP (and MSP) which only requires shunting either on arrival or at departure. Without loss of generality, in this section, we consider only shunting movements on arrival.

An optimal solution of TDP satisfies the following property. Let π_X be the optimal assignment of trams to positions. Then, for each type $\tau \in \mathcal{T}$ and each stack $\mathcal{P}_r$ the following holds:

Lemma 4.4.1: Let π_X be an optimal assignment of trams to stack positions. If $t(a_i) = t(a_j) = \tau$, $i < j$, and $\pi_X(a_i), \pi_X(a_j) \in \mathcal{P}_r$, then the position $\pi_X(a_i)$ is below position $\pi_X(a_j)$.

Proof: We assume that there is a pair of trams $a_i, a_j \in \mathcal{A}$ for which the lemma does not hold. Hence, a_i and a_j have to be shunted since $p_q = \pi_X(a_i)$ is on top of $p_l = \pi_X(a_j)$.

We define a new assignment π_X^* in the following way:

$$\pi_X^*(a_i) = \pi_X(a_j), \quad \pi^*(a_j) = \pi_X^*(a_i), \quad \pi_X^*(a_k) = \pi_X(a_k) \text{ for all } a_k \in \mathcal{A}, k \neq i, j$$

Since $t(a_i) = t(a_j)$, π_X^* is type-preserving. According to π_X^*, a_i and a_j do not have to be shunted. Additionally, the number of shunting movements for the other trams do not change. Let $\mathcal{A}_1$ denote the set of trams $a_k \in \mathcal{A}$ with $k < i$, $\mathcal{A}_2$ be the set of trams $a_k \in \mathcal{A}$ with $i < k < j$, and $\mathcal{A}_3$ be the set of trams $a_k \in \mathcal{A}$ with $k > j$. For π_X and π_X^*, there are the same shunting movements between trams in $\mathcal{A}_\mu$ and $\mathcal{A}_\nu$, $\mu, \nu \in \{1, 2, 3\}$. Trams in $\mathcal{A}_1$ and $\mathcal{A}_3$ that have to be shunted with both trams a_i and a_j in accordance with π_X must also be shunted with both trams if there are assigned according to π_X^* (cf. Figure 4.4.7).

The trams in $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ that are to be shunted with exactly one tram of a_i and a_j when assigned with respect to π_X still have to be shunted with exactly one tram when they are assigned due to π_X^*.

Consequently, π_X^* requires less shunting movements than π_X which contradicts the assumption that π_X is optimal. ❏

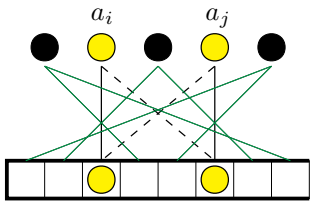

Figure 4.4.7: Switching the assignment for trams of the same type.

For MSP, we can force to consider only such assignments that satisfy the above lemma by adding the following set of constraints:

$$x_{iq} + x_{jl} \leq 1 \text{ for all } a_i, a_j \in \mathcal{A}, i < j,\ p_q, p_l \in \mathcal{P}_r,\ \text{for some} r, q > l. \qquad (4.4.48)$$

We combine the linearization of Kaufman and Broeckx with these additional constraints and denote the resulting integer program by LATDP3. For test instances which are generated from real-world instances of the Braunschweig storage yard for trams, the Magdeburg storage yard, and the bus depot of Wolfsburg, we observe the following performance of LATDP1, LATDP2, and LATDP3 (cf. Table 4.1): the linearization method of Kaufman and Broeckx yields significantly better computation times than the linearization method of Frieze and Yadegar. We can often improve the performance of LATDP2 by adding the additional constraints (4.4.48).

As a consequence, we will solve MSP using the linearization method of Kaufman and Broeckx and adding the additional constraints (4.4.48). The resulting model LADP has already been presented in the Section 4.1.3.

We solve the LADP for real-world instance as well as for randomly generated test instances applying the CPLEX 6.5 MIP-solver on a Pentium-II (350 MHz) PC with 256 MByte core memory. We observe that even for a small number of less than 30 trams it takes up to 50 minutes to compute the optimal solution for TDP. For larger instances, e.g., of the Karlsruhe storage yard, we reach the time limit of four hours computation time (cf. Table 4.2). A solution for the smallest instance of Braunschweig is presented in Figure 4.4.8.

The random test instances are generated as follows. For the instances, we choose the (uniform) stack length, the number of trams, and the number of types. Beginning with the first arrival, the tram type is chosen uniformly at random. Then, we choose uniformly at random the index of a departure of the same type among all departures that have not been chosen yet.

For the random test instances of ten trams and stack length five, we observe that we can solve the test instances within a few seconds. For the larger instances of fifteen trams and three stacks of length five, we achieve that it sometimes takes several hours until a solution has been proven to be optimal. Note that the

instance	LATDP1	LATDP2	LATDP3	vars	constr.	nonzeros
bs.10	8.71	0.39	0.38	275	497	1780
bs.11	22.60	0.50	0.63	336	807	2786
bs.12	13.64	1.49	0.58	276	574	1977
bs.13	0.63	0.13	0.14	85	120	401
bs.14	22.22	0.34	0.52	299	902	2872
bs.15	6.33	0.34	0.52	290	526	1828
bs.16	13.91	0.47	1.06	294	809	2582
md.21	1336.45	55.63	32.91	656	4583	14237
md.22	-	-	-	607	4137	13002
md.24	1592.96	53.27	78.31	747	6005	18477
wob.01	103.66	609.88	19.17	626	1622	5735
wob.02	266.96	5.24	4.01	590	1432	5251
wob.03	110.18	7.59	20.68	625	1460	5295

Table 4.1: Computation times for the different linearization methods applied to real-world instances. CPU-times obtained by applying the CPLEX MIP Solver 5.0 on a Hewlett Packard 9000-735/125MHz workstation with 144 MByte core memory.

instance	N	R	variables	constraints	nonzeros	SM	CPU sec	LIFO
bs.mo-do	27	9	1872	10062	33381	0	2857	0
bs.fr	27	9	1872	10062	33381	0	2727	0
bs.sa	27	9	1872	10062	33381	0	2798	0
bs.so	14	5	483	1149	4355	0	1	0
ka.26	46	15	5424	60896	182744	(5)	$>$ 14400	0
ka.27	23	7	1311	6704	22738	0	945	0
ka.28	44	15	4939	47472	145963	(1)	$>$ 14400	0
ka.29	46	15	5422	60894	182737	(8)	$>$ 14400	0

Table 4.2: Computational results for solving real-world instance of storage yards in Braunschweig and in Karlsruhe using CPLEX 6.5 MIP-solver on a Pentium-II (350 MHz) PC. SM denotes the number of shunting movements.

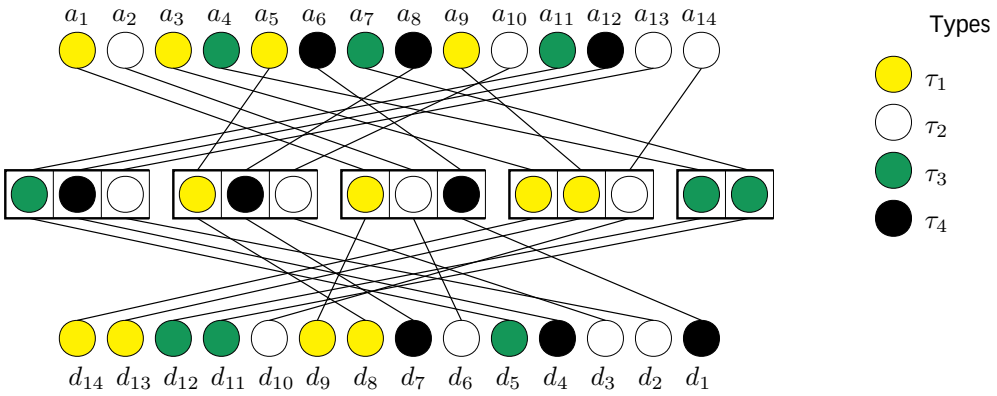

Figure 4.4.8: Solution for the smallest instance of Braunschweig (bs.so).

heuristic LIFO which will be introduced in the next section already yields good results (cf. Table 4.3 – Table 4.8).

10 trams, 2 stacks of length 5								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	3.74	1	0	5.18	1	1	11.01	3
0	9.98	2	2	5.64	2	2	7.23	2
0	0.01	0	1	4.02	1	3	18.20	4
2	21.81	6	0	0.01	0	4	20.27	7
2	35.95	2	2	17.91	2	4	39.09	4
1	25.99	2	1	4.52	1	1	6.81	2
2	8.34	7	3	27.81	4	2	11.52	6
1	8.87	1	2	9.79	4	2	8.94	4
0	14.82	5	1	8.13	7	2	8.16	6
1	15.20	1	1	10.76	1	0	7.01	1

Table 4.3: Computational results for LADP: CPLEX 6.5 applied to random test instances of ten trams, stacks of length five, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

15 trams, 3 stacks of length 5								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	253.92	1	0	3923.70	1	2	14157.11	9
0	0.01	0	0	0.01	0	3	78532.63	9
0	0.01	0	0	0.01	0	0	0.01	0
0	1095.91	2	0	591.68	1	0	3838.22	1
0	2048.14	1	0	425.00	1	0	445.34	1
0	1001.20	3	0	0.01	0	1	1948.24	3
0	393.20	1	0	0.01	0	0	313.18	1
0	395.20	1	0	656.31	2	0	0.01	0
0	548.97	4	0	420.06	2	(2)	$>$ 600000.00	2
0	0.01	0	0	8208.79	1	1	14397.00	3

Table 4.4: Computational results for LADP: CPLEX 6.5 applied to random test instances of fifteen trams, stacks of length five, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

15 trams, 5 stacks of length 3								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	0.01	0	0	0.01	0	0	69.18	2
0	0.01	0	0	0.01	0	0	151.49	1
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	82.44	1
0	1647.03	1	0	0.01	0	0	66.51	1
0	12.21	1	0	0.01	0	0	0.01	0
0	0.01	0	0	29.18	2	0	50.30	1
0	0.01	0	0	0.01	0	0	0.01	0
0	51.20	2	0	68.99	1	0	408.84	2
0	69.52	1	0	0.01	0	0	0.01	0

Table 4.5: Computational results for LADP: CPLEX 5.0 applied to random test instances of fifteen trams, stacks of length three, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

16 trams, 4 stacks of length 4								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	0.01	0	0	0.01	0	0	173.02	1
0	1003.91	1	0	0.01	0	0	0.01	0
0	0.01	0	0	341.41	2	0	1608.08	2
0	0.01	0	0	0.01	0	0	416.10	2
0	0.01	0	0	0.01	0	0	378.77	1
0	0.01	0	0	0.01	0	0	0.01	0
0	159.64	1	0	0.01	0	0	453.09	2
0	177.10	1	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0

Table 4.6: Computational results for LADP: CPLEX 6.5 applied to random test instances of sixteen trams, stacks of length four, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

20 trams, 5 stacks of length 4								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	1098.05	1	0	3038.85	1	0	1589.58	2
0	8125.61	1	0	16289.72	2	-	> 30000.00	1
0	0.01	0	0	61.61	1	(6)	> 7000.00	6
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	7837.89	1	0	0.01	0	0	1763.33	1
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	931.96	1

Table 4.7: Computational results for LADP: CPLEX 6.5 applied to random test instances of fifteen trams, stacks of length three, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

24 trams, 6 stacks of length 4								
3 types			4 types			5 types		
SM	CPU sec	LIFO	SM	CPU sec	LIFO	SM	CPU sec	LIFO
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	-	> 50000.00	1
0	60279.93	1	0	45919.78	1	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	0.01	0	0	0.01	0
0	0.01	0	0	8197.99	1	0	0.01	0
0	0.01	0	0	0.01	0	0	> 60000.00	1
0	0.01	0	0	0.01	0	0	18918.50	1
0	0.01	0	0	84158.93	2	0	8128.54	4
0	11689.26	1	0	0.01	0	0	0.01	0

Table 4.8: Computational results for LADP: CPLEX 6.5 applied to random test instances of fifteen trams, stacks of length three, and three to five types. SM denotes the optimal number of shunting movements. LIFO denotes a first upper bound that the heuristic LIFO yields.

4.5 Heuristics

In the last section, we observed that determining exact solutions for TDP requires large computational times which increase drastically in the number of trams. In this section, we introduce two heuristics for TDP and compare the solutions obtained by applying these heuristics to random and real-world data.

4.5.1 Last-In-First-Out-Heuristic

The first heuristic uses an idea which seems to be promising for stack problems: the **last-in-first-out** principle. This principle may also be considered as a first-in-last-out strategy and works as follows: We always assign the actual arriving tram of type τ to the last unassigned departure of the same type. This assignment is made on arrival. In the following, we identify the trams by the departure assigned to it.

Beside this assignment of arriving trams to departures, we have to search for a stack position at which the tram shall be stored. For the choice of this stack position, we have different possibilities. Since we are interested in assigning the arriving trams to the positions without shunting, we have (at most) R different positions where R denotes the number of stacks. These positions are the free positions on top of the current top positions of the stacks with free positions, i.e., the top-most positions to which no tram has been assigned yet. Such a stack is called **open**. We call a stack r **closed** if a tram has been assigned to position P_r in this stack, $1 \leq r \leq R$.

If we assume that there are already trams assigned to some stacks, we may decide to assign the actual arriving tram assigned to departure $d_j \in \mathcal{D}$ to

1. an open stack where the tram at the current top position is assigned to a departure d_k with $k > j$

2. an open stack where the tram at the current top position is assigned to a departure d_k with $j > k$

3. an empty stack (if such a stack exists)

The current top position in a stack is the top-most position to which a tram has been assigned. We refer to the corresponding tram as the tram on top if this stack. A shunting movement with this tram is required for an assignment of the second type.

LIFO (cf. Definition 4.5.2) tries to avoid shunting as long as possible. In the case that there is an empty stack and a stack to which the actual tram may be assigned without shunting, we have two possibilities for such shunting-free assignments of this tram. If the indices of the actual tram's departure d_i and of the departure d_k of the tram at the current top position in stack r_1 differ

significantly, the assignment of the actual tram to this stack may waste space for other trams assigned to a departure that takes place between d_i and d_k. By the acceptance parameter Δ, we decide in which cases (for which differences between the indices) an empty stack shall be preferred.

Based on these notations, we define the main loop of **last-in-first-out-heuristic (LIFO)** as follows. This main loop will be iterated for each arriving tram.

Last-in-first-out Heuristic (LIFO) 4.5.2:

Assign the actual arriving tram $a_i \in \mathcal{A}$ to the unassigned departure $d_j \in \mathcal{D}$ of the same type ($t(a_i) = t(d_j)$) having maximum index.

Let r_1 be the open stack with a tram assigned to a departure d_k on top where $k > j$ and k is minimal.
Let r_2 be the open stack with a tram assigned to a departure d_l on top where $l < j$ and l is maximal.
Let r_3 be an empty stack.
Let Δ be an acceptance parameter.

if an open stack r_1 exists **then**
  if $k - j \leq \Delta$ **or** no empty stack r_3 exists **then**
    assign a_i to the corresponding position in stack r_1
  else
    assign a_i to the corresponding position in stack r_3
if no open stack r_1 exist **and** there is an empty stack r_3 **then**
  assign a_i to the corresponding position in stack r_3
if no open stack r_1 exists **and** no empty stack r_3 exists **then**
  assign a_i to the corresponding position in stack r_2

■

Next, we present some instances for which LIFO produces a suboptimal solution.

Lemma 4.5.3: LIFO is not optimal.

Proof: We prove the lemma by giving an instance for which LIFO needs one shunting movement whereas the optimal solution does not require shunting. This instance is illustrated in Figure 4.5.9.

We consider the arrival of four trams a_1, a_2, a_3, and a_4. Tram a_1 and a_3 are of type τ_1, a_2 is of type τ_2, and a_4 has type τ_3. The four corresponding departures are of the following types: d_1 requires type τ_2, d_2 and d_4 have type τ_1, and d_4 is of type τ_3. The depot consists of two stacks of length two.

LIFO assigns a_1 to departure d_4 and a_3 to d_2. The assignment of a_2 and a_4 is fixed by the type constraints such that a_2 is assigned to d_1 and a_4 is assigned to d_3.

Without loss of generality, a_1 is assigned by LIFO to the bottom position of stack 1. The assignment of the second tram a_2 depends on the parameter Δ. LIFO accepts differences of Δ departures times.

Case 1: $\Delta \geq 3$: LIFO assigns a_2 to stack 1. The remaining trams a_3 and a_4 are assigned to stack 2 in the order of their arrival. We achieve an assignment that requires one shunting movement for tram a_3 and a_4, since a_3 has to leave earlier than a_4.

Case 2: $\Delta \leq 2$: LIFO chooses an empty stack and assigns a_2 to stack 2. Tram a_3 which is assigned to d_2 is assigned to stack 1. The last tram a_4 is assigned to stack 2. Once again, we achieve a shunting movement for stack 2.

OPT: An optimal algorithm would for instance assign the four trams to the stacks beginning with stack 1. Then, a_1 is assigned to d_2 and a_3 is assigned to d_4. This assignment does not require shunting.

Consequently, LIFO is suboptimal.

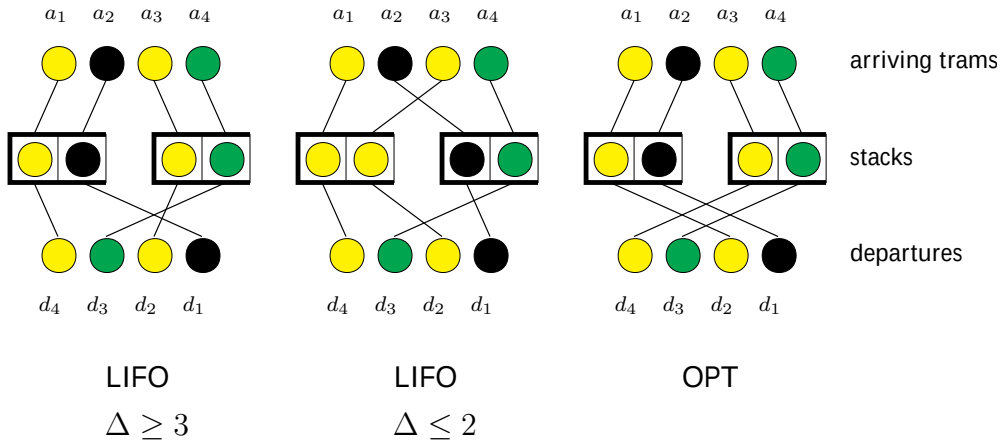

Figure 4.5.9: Counterexample for the optimality of LIFO.

❑

A more general class of instances is given in the following theorem.

Theorem 4.5.4: LIFO needs $2\lfloor \frac{N}{4} \rfloor$ shunting movements for a class of instances of N trams for which an optimal algorithm yields a shunting-free and type-preserving solution.

Proof: We define the class of instances as follows. We consider $N = 6m$ trams of three types τ_1, τ_2, τ_3, $m \geq 1$. The arrival sequence $\mathcal{A}$ is given by the following sequence of types starting with a_1

$$(\tau_1, \tau_2, \tau_3, \ldots, \tau_1, \tau_2, \tau_3)$$

where the partial sequence τ_1, τ_2, τ_3 is repeated $2m$ times. The departure sequence $\mathcal{D}$ is given by the same sequence of types starting with d_1. The depot consists of two stacks, each of length $3m$.

An optimal algorithm assigns the trams to the stacks in the following way. For each position, we give the type of the assigned tram (beginning with the bottom positions):

$$\begin{array}{ll} \text{stack 1} & (\tau_1, \tau_2, \tau_1, \tau_1, \tau_2, \tau_1, \ldots, \tau_1, \tau_2, \tau_1) \\ \text{stack 2} & (\tau_3, \tau_2, \tau_3, \tau_3, \tau_2, \tau_3, \ldots, \tau_3, \tau_2, \tau_3) \end{array}$$

Obviously, no shunting movement is required on arrival or at departure. The trams of type τ_1 are always assigned to the first stack, all trams of type τ_3 are assigned to the second stack, and every second tram of type τ_2 is assigned to the second stack. The same holds for the departures.

For arbitrary acceptance parameter Δ, LIFO assigns the trams as follows (cf. Figure 4.5.10) LIFO starts with assigning the first tram of type τ_1 to the first stack. The second tram, having type τ_2, has to leave the depot later than the first. This tram is assigned to stack 2, because there is no stack r_1 but an empty stack r_3. For the assignment of the third tram of type τ_3, there is no stack r_1 and no empty stack r_3. Consequently, this tram is also assigned to the second stack. LIFO continues with assigning the trams of type τ_1 to stack 1 and the trams of type τ_2 and τ_3 to stack 2 until stack 2 is filled completely. Then, the remaining trams are assigned to stack 1.

LIFO yields an assignment of trams to stack positions as follows (see also Figure 4.5.10):

$$\begin{array}{lccccccc} & a_1 & a_4 & & \ldots & & a_{6m-2} & a_{6m-1} & a_{6m} \\ \text{stack 1} & (\tau_1, & \tau_1, & \ldots, \tau_1, \tau_3, \tau_1, \tau_2, \tau_3, \ldots, & & & \tau_1, & \tau_2, & \tau_3) \\ & d_{6m-2} & d_{6m-5} & & \ldots & & d_1 & d_2 & d_3 \end{array}$$

$$\begin{array}{lcccccc} & a_2 & a_3 & a_5 & a_6 & & \\ \text{stack 2} & (\tau_2, & \tau_3, & \tau_2, & \tau_3, & \ldots, \tau_2, \tau_3, \ldots, \tau_2, \tau_3, \tau_2) & \\ & d_{6m-1} & d_{6m} & d_{6m-4} & d_{6m-3} & & \end{array}$$

if m is odd. Otherwise:

$$\begin{array}{lccccccc} & a_1 & a_4 & & \ldots & & a_{6m-2} & a_{6m-1} & a_{6m} \\ \text{stack 1} & (\tau_1, & \tau_1, & \ldots, \tau_1, \tau_1, \tau_2, \tau_3, \ldots, & & & \tau_1, & \tau_2, & \tau_3) \\ & d_{6m-2} & d_{6m-5} & & \ldots & & d_1 & d_2 & d_3 \end{array}$$

$$\begin{array}{lcccccc} & a_2 & a_3 & a_5 & a_6 & & \\ \text{stack 2} & (\tau_2, & \tau_3, & \tau_2, & \tau_3, & \ldots, \tau_2, \tau_3, \ldots, \tau_2, \tau_3, \tau_2, \tau_3) & \\ & d_{6m-1} & d_{6m} & d_{6m-4} & d_{6m-3} & & \end{array}$$

For each stack, LIFO needs $\lfloor \frac{3m}{2} \rfloor$ shunting movements. In stack 2, every tram of type τ_3 on top of a tram of type τ_2 has to leave the depot later than this tram of type τ_2. Since there are $\lfloor \frac{3m}{2} \rfloor$ such trams, we obtain the same number of shunting movements. In stack 1, there are $\lfloor \frac{m}{2} \rfloor$ triples (τ_1, τ_2, τ_3) that have to

leave in reverse order resulting in 3 shunting movements for each triple. If m is odd, we have one additional shunting movement for the tram of type τ_3 on top of the pure sequence of trams of type τ_1. This results in $3\lfloor\frac{m}{2}\rfloor$, if m is even, and in $3\lfloor\frac{m}{2}\rfloor + 1$, if m is odd. In both cases, the number of shunting movements is equal to $\lfloor\frac{3m}{2}\rfloor$ leading to $\lfloor\frac{3m}{2}\rfloor = \lfloor\frac{N}{4}\rfloor$ shunting movements for each stack. ❑

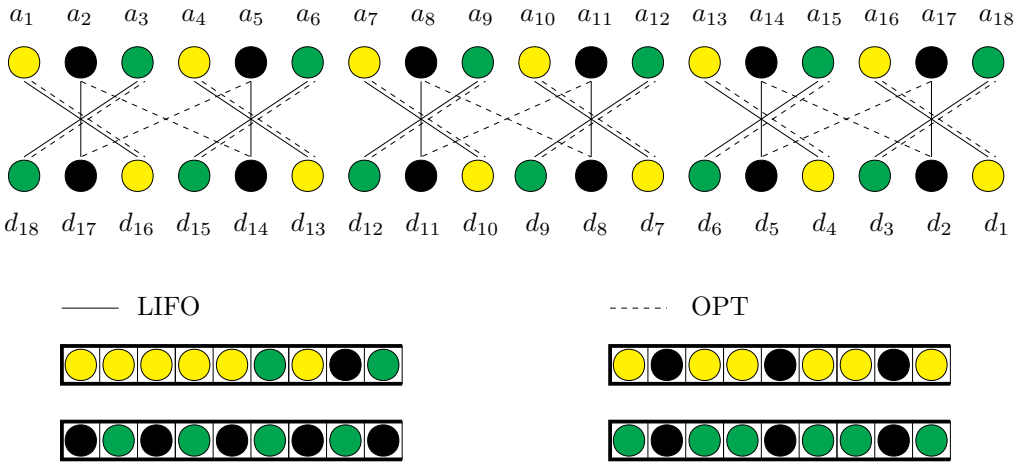

Figure 4.5.10: The solutions obtained by LIFO and OPT for $m = 3$.

Since LIFO assigns each arriving tram without knowledge about the remaining arrivals, LIFO is an online algorithm for TDP. In Section 7.2, we will examine the performance of online algorithms for TDP. By Theorem 4.5.4, LIFO is shown not to be (c, d)-competitive where $d < 2\lfloor\frac{N}{4}\rfloor$. The notion (c, d)-competitive is introduced in Chapter 7.

trams	3 types	max	5 types	max	7 types	max	9 types	max
20	1.28	8	2.44	9	4.26	16	5.03	21
30	0.54	7	1.36	10	2.54	12	3.55	15
40	0.65	5	1.00	9	1.59	7	2.57	9
50	0.36	9	1.25	12	1.49	8	1.91	9

Table 4.9: Average number of shunting movements for 100 random instances with stacks of length five and trams of three to nine types. "max" denotes the maximum number of shunting movements required for one of the 100 instances.

A slightly modified LIFO heuristic, called **LIFO2**, works as follows. In the case that shunting is required and no empty stack exists, LIFO2 assigns tram a_i to the stack r_2 for which the departure of the current top element has the

largest distance to d_j. However, the worst case examples of Lemma 4.5.3 and Theorem 4.5.4 hold for LIFO2, too.

4.5.2 Best-Fit-Heuristic

An alternative approach for a heuristic for TDP is presented by the **best-fit-heuristic (BF)**. In contrast to LIFO, the best-fit-heuristic starts with searching a possible position for the actual arriving tram before a departure of suitable type is assigned. The following main loop of the best-fit-heuristic is iterated for all arriving trams.

Best-Fit-Heuristic (BF) 4.5.5:

Let $a_i \in \mathcal{A}$ be the actual arriving tram.

Let r_1 be an open stack with tram $a_{i(r_1)}$ assigned to departure $d_{j(r_1)}$ at the current top position. r_1 is chosen in such a way that there is an unassigned departure d_j of type $t(a_i)$ with $j < j(r_1)$. If there is such a stack and such a departure, then the maximum index j is chosen.

Let r_2 be an open stack with tram $a_{i(r_2)}$ assigned to departure $d_{j(r_2)}$ at the current top position. r_2 is chosen in such a way that there is an unassigned departure d_k of type $t(a_i)$ with $k > j(r_2)$. If there is such a stack and such a departure, then the minimum index k is chosen.

Let r_3 be an empty stack (if such a stack exists). Let d_m be the unassigned departure having maximum index m.

Let Δ be an acceptance parameter.

if an open stack r_1 exists **then**
  if $j - j(r_1) \leq \Delta$ **or** there is no empty stack r_3 **then**
    assign a_i and d_j to the current position in stack r_1
  else
    assign a_i and d_m to the current position in stack r_3
else if there is an empty stack r_3 **then**
    assign a_i and d_m to the current position in stack r_3
  else
    assign a_i and d_k to the current position in stack r_2

■

LIFO and BF both try to avoid shunting as long as possible. In the case that there is an empty stack and a stack to which the actual tram may be assigned without shunting, we have two possibilities for such shunting-free assignments of this tram. If the indices of the actual tram's departure d_i and of the departure d_k of the tram at the current top position in stack r_1 differ significantly, the assignment of the actual tram to this stack may waste space for other trams assigned

to a departure that takes place between d_i and d_k. By the acceptance parameter Δ, we decide in which cases (for which differences between the indices) an empty stack shall be preferred. The performance of BF and LIFO is illustrated in Figure 4.5.11 for different values of Δ. The heuristics BF+BB and LIFO+BB which apply an a-posteriori reoptimization of π_Y are defined in Definition 4.5.6. We observe that Δ should not be chosen too small. For the following computations, we apply the heuristics for different values of Δ and choose the best solution. Note that LIFO and BF are polynomial-time algorithms which require less than one second computation time for all the instances considered in this thesis.

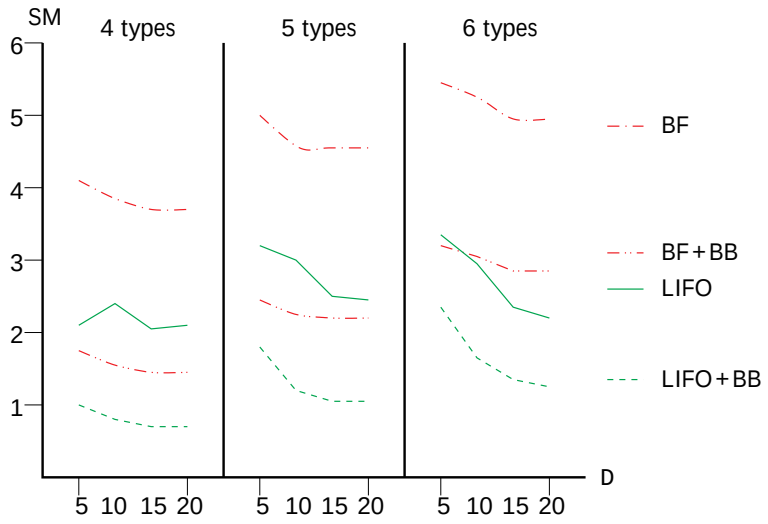

Figure 4.5.11: Average performance of the heuristic methods for different values of the acceptance parameter Δ.

The solutions obtained by LIFO and BF may be improved by solving the corresponding DTDP for the assignment of arriving trams to positions generated by LIFO and BF, respectively. The assignment of departures to the trams stored at the stack positions is recomputed by the enumeration algorithm BB (cf. Definition 5.3.1). We start the calculation of BB using the respective upper bound provided by LIFO or BF. The resulting two heuristics can be considered as **preplan heuristics**.

A preplan heuristic is motivated by the obvious decomposition of TDP. At first, we choose a shunting-free assignment π_X of trams to stack positions which fixes a type pattern for the positions in the stacks. For each position $p_q \in \mathcal{P}$, we define the type of p_q as $t(p_q) = t(\pi_X^{-1}(p_q))$. In the second step, we compute a type-preserving assignment π_Y^* matching to this type pattern and minimizing shunting at departure. The resulting problem instance is an DTDP instance. We will discuss computational results for DTDP in Chapter 5.

We make use of the idea of the preplan heuristic when improving the solutions obtained by the following three heuristics (cf. Algorithm 5.3.1).

LIFO+BB / LIFO2+BB / BF+BB 4.5.6:

> Based on a solution for TDP computed by LIFO, LIFO2, or BF, we solve the corresponding DTDP for given assignment of trams to positions π_X by the enumeration algorithm BB.

■

The enumeration algorithm BB is implemented analogously to the dynamic programming approach introduced in Chapter 3. The enumeration algorithm has exponential worst-case performance. It should therefore only be used for small instances of about 30 trams.

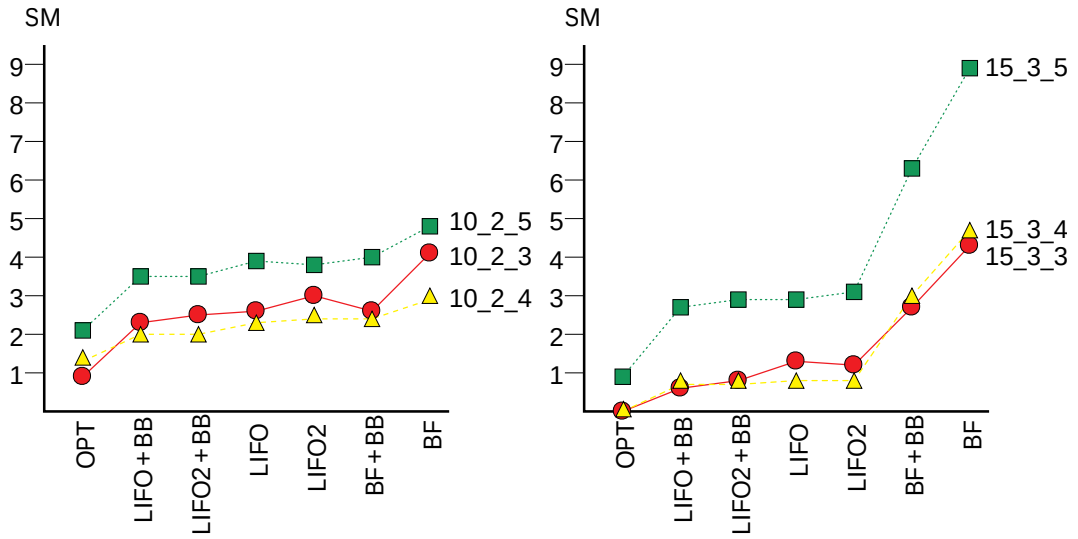

Figure 4.5.12: Average number of shunting movements for ten random instances of size 10_2_T and 15_3_T for $T = 3, 4, 5$ where N_R_T denotes the number of trams, stacks, and types.

Figure 4.5.12 compares the average upper bounds derived by the different heuristics. The average is calculated for ten random instances of the respective sizes (10_2_T and 15_3_T with $T = 3, 4, 5$). We make several runs for different values of Δ and choose the best result. Since no significant improvement can be achieved by the expensive enumeration step, we consider LIFO as the overall best heuristic.

4.6 Minimizing the Number of Type Mismatches

For the real-world instances of Braunschweig and Karlsruhe introduced above, we observe the following results for the corresponding binary linear program MTMP applying the CPLEX 6.5 MIP solver (cf. Table 4.10). We can only solve the smallest instance whereas for all other instances we reached the time limit of 18000 CPU seconds on a Pentium-II PC (350 MHz, 256 MB). For all instances, the upper bound provided by LIFO is zero and hence LIFO yields optimal solutions.

instance	variables	constraints	nonzeros	TM	CPU sec
bs.mo-do	9846	18847	57771	(7)	> 18000
bs.so	1384	1951	6689	0	400
bs.fr	9846	18847	57771	(10)	> 18000
bs.sa	9918	18883	57879	(9)	> 18000
ka.26	75740	120717	394562	-	> 18000
ka.27	7580	12738	40969	(6)	> 18000
ka.28	57044	96331	308390	-	> 18000
ka.29	74306	120000	390902	-	> 18000

Table 4.10: TMP: Results for storage yards at Braunschweig and Karlsruhe.

We also solve the TMP-instances corresponding to the previously defined random test instances of TDP. At first, we derive an upper bound (UB) on the minimum number (TM) of type mismatches from the upper bound of the LIFO heuristic (by applying Theorem 4.3.13). Using this bound, we apply the CPLEX 6.5 MIP solver on a Pentium II PC with 350 MHz and 256 MByte core memory.

10 trams, 2 stacks of length 5								
3 types			4 types			5 types		
TM	UB	CPU sec	TM	UB	CPU sec	TM	UB	CPU sec
0	2	4.48	0	2	6.83	2	4	23.44
0	2	4.86	2	2	273.51	2	3	271.97
0	0	0.01	2	2	90.85	3	4	88.87
2	6	57.85	0	0	0.01	2	6	10.33
2	2	144.94	2	3	104.21	2	4	21.07
2	2	148.41	2	2	439.98	2	2	73.28
2	6	25.39	2	2	64.35	2	4	27.71
2	2	177.52	2	4	60.14	2	2	191.11
0	6	36.79	2	6	50.34	2	6	8.73
2	3	282.67	2	2	165.19	0	2	4.02

Table 4.11: TMP: Solving random test instances.

The computation times for solving MTMP are significantly larger than those for solving the corresponding LADP. Several instances with fifteen trams and three stacks are not solved even within a five hours' time limit. For the instances with ten trams, the optimal shunting-free solutions require at most three type mismatches.

4.7 Conclusion

In Chapter 3, the tram dispatch problems TDP and TMP are shown to be $\mathcal{NP}$-hard which implies that there is no polynomial-time algorithm known for these problems and it seems to be very unlikely that such an algorithm exists. The computational results for random as well as for real-world instances show that the computation times for the exact methods increase drastically with the number of trams to be dispatched. Computing an optimal solution for LADP (see page 72) needs significantly less time than solving MTMP (see page 77). Good solutions for TMP of at most twice the number of type mismatches can be achieved by first solving the corresponding TDP and secondly applying Theorem 4.3.13.

We observe that the real-world instances often admit a shunting-free and type-preserving solution. For the real-world instances considered in this thesis, such a solution is also computed by the LIFO heuristic. This heuristic yields near-optimal solution for the considered random instances, too. If sufficient computation time is available, the solutions obtained by LIFO may be improved by solving the resulting DTDP instance using the enumeration algorithm BB (cf. Definition 5.3.1).

In conclusion, the exact methods can be applied to real-world instances of up to 30 trams. If a solution is required within a short time interval, we recommend the use of LIFO for different values of the acceptance parameter Δ.

Chapter 5

Dispatch of Trams to Departures

In this chapter, we concentrate on the problem of dispatching trams to departures. We assume that all trams have arrived at the depot and are assigned to depot positions. The problem is to assign trams to the departures corresponding to the round trips of the next schedule period. In particular, such a situation occurs in the early morning when the dispatcher has to assign already stored trams to the next round trips. Blasum et al. [BBH+98] examine the decision version of this problem where we are interested in a shunting-free and type-preserving solution. We consider the two variations of the departure dispatch problem: the minimum shunting problem (DTDP) (cf. Section 3.4) and the minimum type mismatch problem (DTMP) (cf. Section 3.5).

5.1 Minimizing Shunting at Departure

For each departure, the dispatcher has to choose a tram of suitable type to satisfy the requirements given by the corresponding round trip and the schedule. The trams are assumed to be already stored in the depot. By π_X, we denote the assignment of trams to depot positions. Given this assignment, the dispatcher has to assign the trams to the departures. The objective is to minimize the number of shunting movements necessary to let the trams leave the depot.

5.1.1 An Integer Program for the Departure Problem

In Chapter 4, we derived a quadratic assignment model for the departure problem DTDP. In this problem, we are given a fixed assignment of trams to depot positions. The objective is to find an assignment of stored trams of suitable type to the departures minimizing the number of shunting movements necessary for the trams' departure.

In this section, we assume that the number of trams is equal to the number of positions and to the number of departures. In Section 5.5, we will consider

the case where not all trams have to leave the depot for practical data. The type of a tram stored at position p_q is defined to be the type of the arriving tram which has been assigned to this position. We define $t(p_q) = t(a_i)$ for $a_i \in \mathcal{A}$ and $p_q \in \mathcal{P}$ and $\pi_X(a_i) = p_q$ (or, in other words, if the corresponding coefficient of the assignment matrix X_{iq} equals to 1 (cf. Chapter 4)).

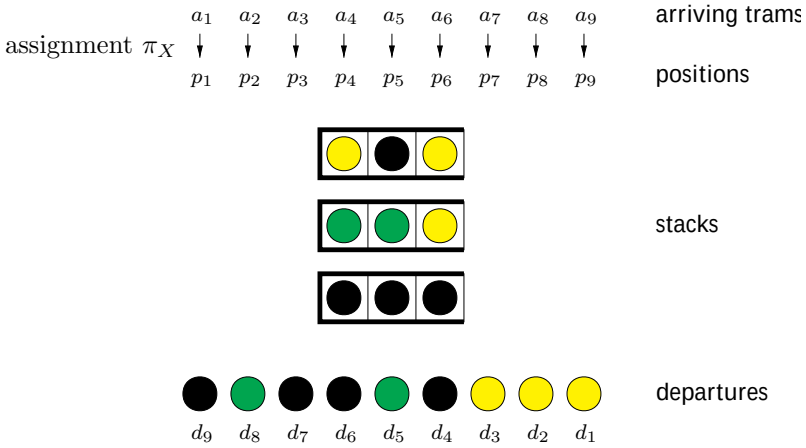

Figure 5.1.1: An instance of the departure problem.

$$\textbf{(DP)} \quad \min \quad \sum_{d_i \in \mathcal{D}} \sum_{d_k \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} \alpha_{ik} \beta_{lq} \, y_{iq} \, y_{kl} \tag{4.1.28}$$

$$\text{s.t.} \quad \sum_{d_i \in \mathcal{D}} y_{iq} = 1 \qquad \text{for all } p_q \in \mathcal{P} \tag{4.1.29}$$

$$\sum_{p_q \in \mathcal{P}} y_{iq} = 1 \qquad \text{for all } d_i \in \mathcal{D} \tag{4.1.30}$$

$$y_{iq} \in \{0,1\} \qquad \text{for all } d_i \in \mathcal{D},\ p_q \in \mathcal{P} \text{ and } t(d_i) = t(p_q) \tag{4.1.31}$$

Recall that an optimal solution of (DP) satisfies the following inequality

$$y_{jq} + y_{kl} \leq 1 \text{ for all } d_j, d_k \in \mathcal{D} : j < k, \text{ for all } p_q, p_l \in \mathcal{P}_r : q < l \text{ for some } r \text{ and } t(p_q) = t(p_l) \tag{4.1.32}$$

which has been introduced as set of valid inequalities in Chapter 4.

In an optimal solution the following property holds: If two trams of the same type are stored in the same stack and are assigned to two departures, the top-most tram of both is assigned to the earlier departure.

5.1.2 A (Linearized) Mixed Integer Program for DTDP

In Section 4.1.2, we introduced a quadratic binary program DP for the departure problem DTDP. To DP, we can apply the linearization methods for quadratic assignment problems introduced in Chapter 4: the linearization method of Frieze and Yadegar and the linearization method of Kaufman and Broeckx. Moreover, we add the inequality constraints (4.1.32). In Section 4.4.1, we observed that the linearization method of Kaufman and Broeckx using the additional constraints (4.1.32), denoted by LATDP3, has the best performance among the considered methods. A difference between ATDP and DTDP is that in DTDP we have already fixed a particular type for each position. Besides this fact, ATDP and DTDP are equivalent problems differing only in the particular sequence (of trams or departures) and in the definition of the situations when shunting is required. In fact, ADTP with a given assignment of types to positions can be considered as the corresponding departure problem for queues instead of stacks.

We obtain the following linearization of DP:

(LDP)

$$\min \quad \sum_{d_i \in \mathcal{D}} \sum_{p_q \in \mathcal{P}} w_{iq} \tag{5.1.1}$$

$$\text{s.t.} \quad \sum_{d_i \in \mathcal{D}} y_{iq} = 1 \qquad \text{for all } p_q \in \mathcal{P} \tag{5.1.2}$$

$$\sum_{p_q \in \mathcal{P}} y_{iq} = 1 \qquad \text{for all } d_i \in \mathcal{D} \tag{5.1.3}$$

$$d_{iq}\, y_{iq} - w_{iq} + \sum_{d_k \in \mathcal{D}} \sum_{p_l \in \mathcal{P}} \alpha_{ik}\beta_{lq}\, y_{kl} \leq d_{iq} \qquad \text{for all } d_i \in \mathcal{D},\ p_q \in \mathcal{P} \tag{5.1.4}$$

$$y_{jq} + y_{kl} \leq 1 \qquad \text{for all } d_j, d_k \in \mathcal{D} : j < k \tag{4.1.32}$$
$$\text{for all } p_q, p_l \in \mathcal{P}_r : q < l$$
$$\text{for some } r \text{ and } t(p_q) = t(p_l)$$

$$y_{iq} \in \{0,1\}, w_{iq} \geq 0 \qquad \text{for all } d_i \in \mathcal{D},\ p_q \in \mathcal{P} \tag{5.1.5}$$

Here, $d_{iq} = \sum_{d_j \in \mathcal{D}} \sum_{p_l \in \mathcal{P}} \alpha_{ij}\beta_{ql}$ and w_{iq} can be interpreted as $w_{iq} = y_{iq} \sum_{d_j \in \mathcal{D}} \sum_{p_l \in \mathcal{P}} \alpha_{ij}\beta_{lq}\, y_{jl}$.

5.2 Shunting-free Solutions

In Section 3.4.1 and in Section 3.5, we introduced two dynamic programming approaches for dispatching trams to departures without shunting. For a fixed number of stacks, we observed that we can decide in polynomial-time whether of not a shunting-free solution exists. Moreover, we have shown that for a fixed number of stacks the type mismatch problem at departure (DTMP) can be solved

in polynomial-time. In Section 5.6, we present computational results for this dynamic programming approach for DTMP.

Next, we will briefly recall the idea of the dynamic programming approach. According to Definition 3.5.33, the state space of the dynamic programming approach is

$$\mathcal{S} := \prod_{i=1}^{R} \{0, 1, \dots, P_r\}$$

where P_r denotes the length of stack r, $1 \leq r \leq R$. Each state of $s \in \mathcal{S}$ is identified by an r-tuple $(s_1, s_2, \dots, s_R)$ where s_r denotes the number of trams that are already assigned to some departures without shunting. Hence, each state $s \in \mathcal{S}$ corresponds to a partial assignment of trams to the first $k(s) := \sum_{i=r}^{R} s_r$ departures.

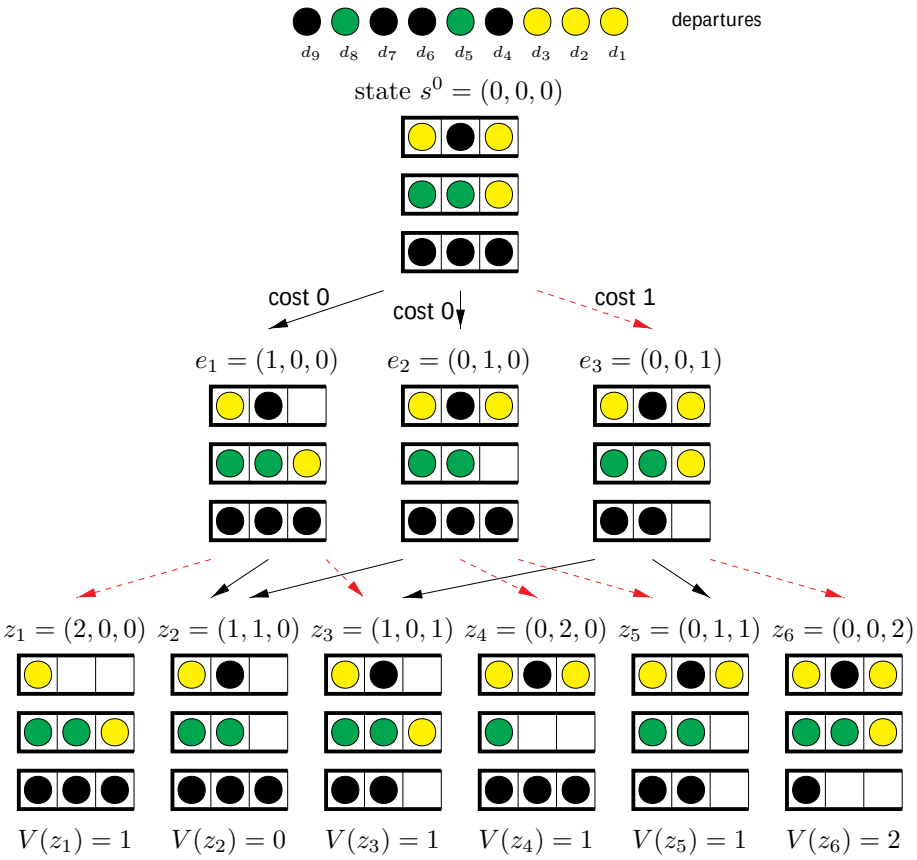

Figure 5.2.2: An example for the dynamic programming approach for 0-DTDP and DTMP.

In Figure 5.2.2, we give an example for the dynamic approach for 0-DTDP and DTMP. In this example, we present the first two steps of the dynamic program-

ming approach. For the considered instance, no shunting-free and type-preserving solution exists. Such an assignment only exists for the first two departures d_1 and d_2. This assignment corresponds to state z_2 for which $V(z_2) = 0$ and to a path from s^0 to z_2 following the solidly printed edges, i.e., either s^0, e_1, z_2 or s^0, e_2, z_2. A state transition step which requires a type mismatch is illustrated by the dashed printed edges between two states. The optimal number of type mismatches is given by $V((3,3,3)) = 2$.

5.3 Enumerating the Solutions of DTDP

For determining the minimum number of shunting movements, we follow a similar approach. Once again, we start with a state representing the situation where no tram has left the depot. From this initial state, we proceed as follows. The first departure of type $\tau \in \mathcal{T}$ has to be assigned to a tram of the same type stored at some position in the depot. Let m be the number of such trams of type τ. Then, we have m possibilities to assign the first departure. With each possibility, we identify a state for which we have to note which trams have already been assigned to which departure.

In Figure 5.3.3, we show how the enumeration algorithm works for the example of Figure 5.2.2. For the first departure d_1, we have three possibilities: There are three trams of type $\tau = t(d_1)$, two in stack $\mathcal{P}_1$ and one in stack $\mathcal{P}_2$. If we decide to assign d_1 either to the tram at the top position of $\mathcal{P}_1$ or to the tram at the top position of stack $\mathcal{P}_2$, this assignment does not require shunting. If we assign d_1 to the tram of type τ at the bottom position of stack $\mathcal{P}_1$, this assignment requires two shunting movements. For the second departure d_2 and each decision made before, we have two possibilities resulting in six partial assignments for (d_1, d_2). For the third departure and for each of the six states, the next assignment is fixed. The fourth departure is of different type $t(d_4)$. For this departure, we have four possibilities.

The number of shunting movements that is required by assigning a departure to a tram is illustrated by the edges connecting the two corresponding states. If the edge is solid, then no shunting is required. If the edge is dashed, then one shunting movement is required. The edge is dotted if the assignment requires two shunting movements.

The constraints (4.1.32) imply that an optimal solution of DTDP satisfies the property that for each stack and each departure always the top-most unassigned tram of suitable type is chosen. Hence, we can restrict ourselves to the states that correspond to such an assignment. In Figure 5.3.3, we marked the states that need not be considered by dashed frames.

As in the dynamic programming approach for DTMP, for each departure and for each stack we have at most one possible tram to which the departure may be assigned. The difference between the dynamic programming approach for

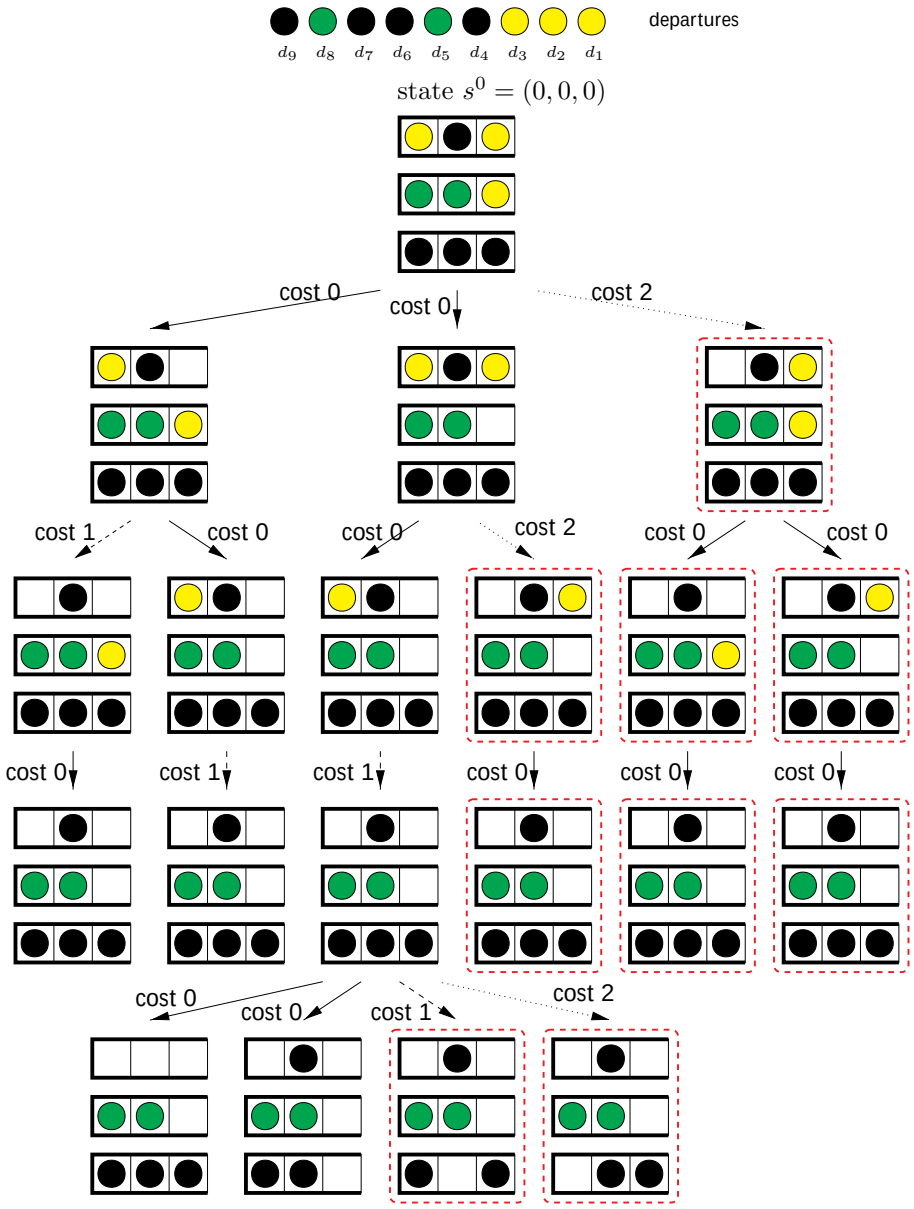

Figure 5.3.3: The enumeration algorithm BB.

DTMP and the enumeration scheme for DTDP is as follows. In our enumeration algorithm, each state corresponds to partial assignment of departures. Each state is identified by the following N-dimensional vector:

$$\rho = (\rho_1, \rho_2, \rho_3, \ldots, \rho_N).$$

where $\rho_j \in \{0, 1, 2, \ldots, R\}$. If $\rho_j \neq 0$, ρ_j denotes the stack to which the departure $d_j \in \mathcal{D}$ is assigned. In particular, d_j is assigned to the top-most position at which

an unassigned tram of suitable type is stored. If $\rho_j = 0$, then departure d_j has not yet been assigned to a position.

We assign the departures step by step beginning with d_1. In each step, if $\rho_j = 0$, then $\rho_k = 0$ for all $k > j$. For each component ρ_j, $1 \leq j \leq N$, we have at most $R+1$ possibilities. Consequently, there are exponentially many states so that the enumeration algorithm is not polynomial.

Algorithm 5.3.1:

By **BB**, we denote the above enumeration algorithm for DTDP.

■

Here, BB stands for **upper bound based enumeration algorithm**, because BB fathoms a partial assignment of departures and all corresponding complete assignments of departures if the solution value of the partial assignment is equal to the value of the current best solution found so far.

5.4 Heuristics

In Chapter 3, we have seen that even deciding whether or not there is a shunting-free and type-preserving assignment of stored trams to the departure is $\mathcal{NP}$-complete. Consequently, the departure problem DTDP is $\mathcal{NP}$-hard and the application of heuristics may be considered. In this section, we present a greedy heuristic, called **GREEDY-DTDP** or briefly **GREEDY**, and a local search heuristic developed by Battiti and Tecchiolli [BT94], called **Reactive Tabu Search (RTS)**. For a comprehensive introduction in modern heuristic techniques, like local search heuristics, we refer to [Ree95].

5.4.1 Neighborhood

Before we present the heuristics for DTDP, we define the notion **neighborhood** between two solutions in the solution space. We identify each solution of DTDP, i.e., the assignment π_Y of departures to (trams at) depot positions, by a N-dimensional vector $\rho = (\rho_1, \rho_2, \ldots, \rho_N)$ where ρ_i denotes the stack to which the departure $d_i \in \mathcal{D} = \{d_1, \ldots, d_N\}$ is assigned. Since an optimal assignment always chooses the top-most unassigned tram of the required type as first, the corresponding stack position is well-defined.

Two solutions $\rho^{\mathbf{1}}$ and $\rho^{\mathbf{2}}$ are said to be **neighbors** if and only if they differ in exactly two components ρ_i and ρ_j, $i \neq j$, where $\rho_i^1 = \rho_j^2$ and $\rho_j^1 = \rho_i^2$. Since both solutions are feasible for DTDP and therefore type-preserving, it follows that $t(d_i) = t(d_j)$. Consequently, two solutions are neighbors if they differ exactly in the way in which they assign two departures of the same type.

5.4.2 A Greedy Heuristic

An natural approach to find a first acceptable solution is the application of a greedy heuristic. GREEDY-DTDP heuristics always chooses iteratively the "best" local improvement. For each departure, GREEDY-DTDP always chooses a tram of suitable type which can be assigned with a minimum number of shunting movements, i.e., with a minimum number trams on top which have not yet been assigned. We call such a tram a **top most unassigned** tram.

GREEDY-DTDP 5.4.2:

> For each departure, GREEDY-DTDP chooses the top-most unassigned tram of suitable type. If there are two or more such trams, GREEDY-DTDP chooses the tram from the stack having the smallest index among those stacks containing these trams.

■

GREEDY-DTDP assigns the departures step by step. Hence, GREEDY-DTDP is an online algorithm (cf. Section 2.4 and Chapter 6). Competitiveness results for GREEDY-DTDP will be considered in Chapter 7).

5.4.3 The Reactive Tabu Search Heuristic

The reactive tabu search heuristic was introduced by Battiti and Tecchiolli [BT94]. In standard tabu search, we start with an initial solution and search for the best solutions in the neighborhood. One of the best solutions is chosen as the next solution. The previous solution is marked to be **tabu** and is never visited again. While searching the neighborhood of the current solution, we do not take into consideration solutions that have been marked to be tabu. These solutions are listed in a tabu list which is usually implemented by a hash table. A major drawback of the standard tabu search is that we often get stuck in a region around a local optimum.

The **reactive tabu search** improves the standard tabu search approach as follows. In order to avoid getting stuck in a local optimum, it allows to visit solutions that has been visited before. Such a step is called **tabu move**. The solutions that have already been visited are stored in the tabu list. For each such solution, the number of tabu moves is counted. If the number of tabu moves becomes too large the actual solution is changed by several random switches. After such an **escape move** the tabu list is deleted and the search is continued.

As initial solution, we use the solution that we obtain by applying the greedy heuristic GREEDY-DTDP. Then, 1000 iterations of the reactive tabu search are applied and the best solution is given as output. We call the corresponding algorithm the **reactive tabu search heuristic (RTS)**.

5.5 Computational Results

We solve the DTDP instances to optimality by applying CPLEX 6.5 to the corresponding LDP and by our enumeration algorithm BB. As heuristical methods, we apply GREEDY-DTDP and RTS. These exact and heuristical methods are applied to random instances as well as to real-world instances.

The real-world instances are based on data from several storage yards in Germany, i.e., Braunschweig, Magdeburg, Wolfsburg, and Halle. The real-world instances consists up 14 to 74 trams which are stored in stacks of length two to eight. Moreover, we consider two different situations. In the first situation, all trams staying in the depot have to be assigned to a departure. In the second situation, there are less departures that have to be served than trams in the depot. All our algorithms and models can be adapted to this situation. The LDP can be modified analogously to the modifications of LADP (cf. Section 4.1.4). BB proceeds departure by departure always choosing the top-most tram of suitable type in some stack. The same holds for GREEDY-DTDP and RTS. Consequently, all three algorithms are also able to produce "partial" assignments of trams to departures.

Practical Data

For the Braunschweig storage yard, we consider eight different instances that arise from real-world data of one week. We divide the trams into five groups of types where a type corresponds to the tram's car type and its year of construction. Up to 27 trams have to leave the depot in the early morning within close time intervals. If we search for an one-to-one assignment of trams to departures, we choose the required number of trams of suitable types from the trams stored in the depot. For the second situation, we use the assignment of trams to positions as they are given by the dispatcher's protocol.

Moreover, we apply our algorithms to four instances of the Magdeburg storage yard for trams. The instances consists of four types of trams. 31 trams have to leave the depot in the morning. These trams are stored on six sidings containing eight to eleven positions. For one given assignment of trams to positions, we serve four different departure sequences which correspond to the schedules of four consecutive days.

For the bus depot of the Wolfsburg company WVG, we instances generated from the data of nine different days. For eight days, we decided to consider two different departure sequences because two busses have to leave the depot at the same time. In the two corresponding departure sequences, we examine the two possibilities sorting both departures. Between 37 and 39 busses have to leave the depot except of the one weekend schedule where only 25 busses have to depart. The busses are stored in up to 20 lanes with two to four positions. The busses are divided into five types.

The two instances for the Halle storage yard for trams are the largest ones considered in this thesis. In these instances, 69 and 74 trams divided into up to 14 types have to leave the depot.

In Table 5.1, we present computational results for the first situation where all trams have to leave the depot. We observe that for almost all instances we achieve a shunting-free and type-preserving solution. Some instances could not be solved by BB whereas all corresponding LDP problems are solved using CPLEX 6.5 MIP solver. GREEDY-DTDP often yields suboptimal solutions whereas for the considered instances RTS always finds an optimal solution within less than 35 seconds computation time. By N, R, and T, we denote the number of trams, the number of stacks, and the number of types in the considered instance.

For the second situation where there are less departures than trams in the depot, we obtain similar results. Table 5.2 shows the computational results for the corresponding test instances. BB fails for several instances because of a time limit of two hours for the computation time. By applying CPLEX 6.5 to the LDP problems, we solve all instances within less than one hour. Once again, we observe that RTS finds optimal solutions within less than two minutes. GREEDY-DTDP needs significantly more shunting movements than required by an optimal solution.

In conclusion, we can solve all the practical instances by applying the algorithms introduced above. We observe that, for all instances considered, RTS yields an optimal solution within a computation time less than two minutes. Hence, this heuristic may find an application in real-world dispatching systems. If we are interested in (provable) optimal solutions for the DTDP instances, either BB can be applied or the corresponding LDP can be solved using CPLEX 6.5. For most of the instances, BB needs less time than applying CPLEX 6.5 but fails for some instances whereas all instances are solved by CPLEX 6.5 within less than one hour.

Random Data

In the following, we examine the performance of BB, GREEDY-DTDP, and RTS as well as the time needed for solving the corresponding LDP using CPLEX 6.5. All the instances considered in this section require a one-to-one correspondence between the trams and departures so that it is assumed that all trams leave the depot.

The random instances are generated as follows. Given the number N of trams and the number R of stacks, we first define the stack lengths. Except of the last stack, the stacks contain the average number of positions, i.e., $\lfloor \frac{N}{R} \rfloor$ positions. The last stack contains the remaining positions so that $P = N$. For each instance, we give the number T of types. Next, for each of the N departures we choose a type uniformly at random from $\{1, \ldots, T\}$. Then, a tram of the same type is assigned to a stack position which is chosen uniformly at random among all

instance	N	R	T	CPLEX	sec.	BB	sec.	GREEDY	RTS	sec.
bs.10.1	27	12	5	0	0.84	0	0.1	0	0	0.1
bs.10.2	27	12	5	0	0.22	0	0.1	2	0	0.1
bs.11	27	12	5	0	2.00	0	0.1	2	0	0.1
bs.12	27	12	5	0	0.44	0	0.1	2	0	0.1
bs.13	14	8	4	0	0.05	0	0.1	1	0	0.1
bs.14	27	12	5	0	0.62	0	0.1	0	0	0.1
bs.15	27	15	5	0	0.24	0	0.1	0	0	0.1
bs.16	27	13	5	0	0.32	0	0.1	0	0	0.1
md.20	31	6	3	1	21.45	1	0.3	6	1	32.8
md.21	31	6	3	0	22.38	0	0.1	0	0	0.1
md.22	31	6	3	4	47.72	4	0.1	7	4	25.7
md.23	31	6	3	3	19.64	3	0.1	3	3	26.8
md.24	31	6	3	0	22.07	0	0.1	3	0	0.1
wob.01.1	37	16	5	0	2.20	0	0.1	1	0	0.1
wob.01.2	37	16	5	0	7.58	0	0.1	1	0	0.1
wob.02.1	37	13	5	0	6.20	0	0.4	4	0	0.2
wob.02.2	37	13	5	0	11.66	0	0.4	4	0	0.1
wob.03.1	38	16	5	0	6.21	0	1.3	1	0	6.7
wob.03.2	38	16	5	0	4.17	0	3.6	1	0	1.3
wob.04.1	38	15	5	0	8.51	0	0.1	3	0	0.1
wob.04.2	38	15	5	0	9.35	0	0.1	3	0	0.1
wob.05	25	10	5	0	0.89	0	0.1	4	0	0.1
wob.08.1	39	16	5	0	0.30	0	0.1	0	0	0.1
wob.08.2	39	16	5	0	11.64	0	0.1	0	0	0.1
wob.09.1	37	16	5	0	3.29	0	1420.0	4	0	0.1
wob.09.2	37	16	5	0	3.19	(3)	> 7200.0	4	0	13.7
wob.10.1	37	15	5	0	2.48	0	0.1	2	0	0.1
wob.10.2	37	15	5	0	1.21	0	0.1	2	0	0.1
wob.11.1	37	16	5	0	4.82	0	0.1	0	0	0.1
wob.11.2	37	16	5	0	3.23	0	0.1	0	0	0.1
ha.13	38	16	5	0	552.25	0	0.5	12	0	1.3
ha.14	38	16	5	0	1117.76	(11)	> 7200.0	8	4	204.0

Table 5.1: Computational results for the DTDP instances of the storage yards in Braunschweig (bs), Halle (ha), and Magdeburg (md) and for a bus depot in Wolfsburg (wob) Situation 1: All trams have to leave the depot.

instance	N	P	R	T	CPLEX	sec.	BB	sec.	GREEDY	RTS	sec.
bs.11	27	56	21	5	0	1.52	0	0.2	1	0	0.1
bs.12	27	56	19	5	0	0.90	0	0.1	0	0	0.1
bs.13	14	54	19	4	0	1.47	0	0.1	1	0	0.1
bs.14	27	55	19	5	0	3.00	0	0.1	0	0	0.1
bs.15	27	54	20	5	0	0.46	0	0.1	0	0	0.1
bs.16	27	53	20	5	0	1.04	0	0.1	0	0	0.1
md.20	31	53	6	4	1	457.34	(1)	> 7200.0	1	1	89.8
md.21	31	53	6	4	0	96.93	0	0.1	0	0	0.1
md.22	31	53	6	4	13	485.57	(15)	> 7200.0	22	13	96.5
md.23	31	53	6	4	6	536.21	6	7.7	17	6	93.8
md.24	31	53	6	4	0	661.30	0	0.3	0	0	0.1
wob.01.1	37	50	19	5	0	12.35	0	0.1	1	0	0.6
wob.01.2	37	50	19	5	0	10.92	0	0.1	1	0	0.6
wob.02.1	37	54	19	5	0	6.16	0	0.1	2	0	0.2
wob.02.2	37	54	19	5	0	7.72	0	0.1	2	0	0.2
wob.03.1	38	55	21	5	0	13.66	(2)	> 7200.0	3	0	0.3
wob.03.2	38	55	21	5	0	15.56	(2)	> 7200.0	3	0	0.3
wob.04.1	38	55	21	5	0	15.51	0	3.8	6	0	0.7
wob.04.2	38	55	21	5	0	13.66	0	3.6	6	0	0.7
wob.05	25	51	18	5	0	6.92	0	0.1	1	0	0.1
wob.08.1	39	54	21	5	0	16.66	0	0.1	0	0	0.1
wob.08.2	39	54	21	5	0	22.62	0	0.1	0	0	0.1
wob.09.1	37	53	20	5	0	3.62	(5)	> 7200.0	3	0	0.4
wob.09.2	37	53	20	5	0	3.70	(5)	> 7200.0	3	0	0.4
wob.10.1	37	52	19	5	0	17.70	0	1.6	2	0	0.2
wob.10.2	37	52	19	5	0	7.55	0	1.0	2	0	0.2
wob.11.1	37	53	21	5	0	11.27	0	0.1	0	0	0.1
wob.11.2	37	53	21	5	0	5.12	0	0.1	0	0	0.1
ha.13	74	125	45	14	0	2370.12	-	> 7200.0	12	0	0.1
ha.14	69	124	47	14	0	1395.47	-	> 7200.0	0	0	0.1

Table 5.2: Computational results for the DTDP instances of the storage yards in Braunschweig (bs), Halle (ha), and Magdeburg (md) and for a bus depot in Wolfsburg (wob). Situation 2: Some trams stay in the depot.

free positions. Consequently, there are as many departures of each type as there are trams having the same type. We identify a class of instances by the given parameters N, R, and T and denote it by N_R_T.

We apply our algorithms to the following classes of instances where each class contains ten instances for the specified parameters: 30_5_T, 30_6_T, 30_7_T, 40_5_T, 40_6_T, and 50_5_T where $T \in \{2, 3, \ldots, 6\}$ (cf. Table 5.3 – Table 5.9).

For the random instances, we observe that BB needs less computation time than solving the corresponding LDP using CPLEX 6.5. For the instances of 30 trams, we did not use upper bounds which we are given by the solutions obtained by RTS and GREEDY-DTDP. We make use of this information for the instances with 40 and 50 trams. In the case that the heuristic solution does not require shunting, we already know that this solution is optimal so that we need not to apply the exact methods. In Table 5.6, Table 5.7, and Table 5.8, this situation is denoted by the symbol * in the column containing the computation times for BB.

For the heuristics GREEDY-DTDP and RTS, we observe the following. GREEDY-DTDP needs significantly more shunting movements than RTS. The instances for which RTS yields an optimal solution are marked by printing the corresponding objective value of RTS in bold letters. We observe that RTS yields optimal solution for about 88 percent of the considered random instances. The computation times for RTS are not greater than 160 seconds even for instances of 50 trams. For some of these instances, the exact methods failed in finding optimal solutions (and proving their optimality).

N_R_T	instance	OPT SM	BB sec.	CPLEX sec.	GREEDY	RTS	sec.
30_5_2	1	0	0.1	261.88	7	**0**	0.8
	2	0	0.1	249.42	4	**0**	0.3
	3	0	0.1	321.10	11	**0**	17.3
	4	0	0.1	6934.99	3	**0**	0.1
	5	0	0.1	186.97	**0**	**0**	0.1
	6	0	0.1	93.02	4	**0**	1.1
	7	0	1.6	187.21	6	**0**	0.9
	8	0	2.7	4381.17	4	**0**	7.6
	9	0	0.1	46.83	**0**	**0**	0.1
	10	0	0.1	83.01	3	**0**	0.1
30_5_3	1	4	0.1	323.37	11	**4**	24.9
	2	**3**	1.4	5150.98	18	5	26.8
	3	4	45.8	18981.31	15	**4**	27.3
	4	1	0.2	524.60	6	**1**	26.0
	5	1	2.2	1062.00	7	**1**	26.7
	6	2	34.2	>140000.00	12	4	28.5
	7	0	0.2	184.60	5	**0**	20.3
	8	3	47.6	> 52250.00	7	**3**	26.9
	9	0	0.1	18.83	6	**0**	5.2
	10	1	0.6	74.92	7	**1**	27.5
30_5_4	1	8	0.1	336.73	15	**8**	17.7
	2	7	0.1	236.89	15	**7**	18.6
	3	5	5.6	781.47	13	**5**	22.3
	4	2	0.1	32.50	6	**2**	20.5
	5	7	60.6	8828.31	18	**7**	21.2
	6	7	13.7	26100.97	15	**7**	21.8
	7	8	60.7	1362.38	16	**8**	20.3
	8	2	3.2	16007.37	9	**2**	22.2
	9	5	0.2	575.62	11	**5**	19.8
	10	4	15.1	274.52	15	**4**	19.8
30_5_5	1	8	2.9	1280.83	16	**8**	15.4
	2	6	1.2	24.44	20	**6**	15.4
	3	4	3.3	414.90	14	**4**	16.8
	4	3	0.2	5.55	7	**3**	15.3
	5	10	15.5	1365.17	13	**10**	16.9
	6	5	7.3	1894.85	6	**5**	16.3
	7	16	37.7	14806.15	23	**16**	18.9
	8	10	13.3	40643.57	12	**10**	21.6
	9	7	0.3	5536.27	9	**7**	18.2
	10	5	6.2	120.79	12	**5**	16.5
30_5_6	1	11	11.3	30.40	19	**11**	13.0
	2	14	13.5	294.66	21	**14**	12.6
	3	13	2.6	101.81	27	**13**	15.3
	4	7	1.2	91.77	11	**7**	13.4
	5	5	0.4	16.83	11	**5**	13.8
	6	5	2.5	116.73	14	**5**	14.2
	7	16	11.7	1749.22	21	**16**	16.5
	8	9	4.9	1666.03	12	**9**	15.5
	9	8	0.1	34.47	12	**8**	13.7
	10	7	5.1	41.40	11	**7**	13.4

Table 5.3: Results for random instances of 30 trams stored in 5 stacks.

N_R_T	instance	OPT SM	BB sec.	CPLEX sec.	GREEDY	RTS	sec.
30_6_2	1	0	0.1	78.79	2	**0**	0.8
	2	0	0.1	44.78	**0**	**0**	0.1
	3	1	0.1	9907.23	11	3	40.5
	4	1	0.1	> 38425.00	14	**1**	41.0
	5	0	0.1	18.46	5	**0**	1.8
	6	0	0.1	125.86	2	**0**	0.1
	7	5	6622.1	> 40900.00	10	**5**	40.5
	8	0	0.1	74.43	6	**0**	0.9
	9	0	0.1	115.93	7	**0**	5.7
	10	0	0.1	58.95	**0**	**0**	0.1
30_6_3	1	0	0.1	15.56	3	**0**	0.1
	2	0	0.1	6.35	5	**0**	0.1
	3	8	7.6	> 107000.00	18	**8**	26.6
	4	1	0.1	55.12	14	**1**	27.0
	5	1	1633.20	2082.43	9	**1**	27.1
	6	2	0.6	201.60	7	**2**	29.9
	7	3	1.3	> 50500.00	7	**3**	30.9
	8	0	0.1	434.22	7	**0**	2.2
	9	2	0.1	14.43	10	**2**	26.6
	10	1	5.0	29352.56	6	**1**	29.9
30_6_4	1	1	0.1	17.60	7	**1**	20.8
	2	2	0.5	18.93	5	**1**	20.1
	3	14	944.90	> 10800.00	17	**14**	21.2
	4	6	3.0	141.64	16	**6**	20.5
	5	3	6.2	186.37	6	**3**	20.2
	6	2	0.6	109.30	7	**2**	20.8
	7	8	24.0	1229.66	10	**8**	19.7
	8	0	0.1	3.93	6	**0**	3.2
	9	3	20.5	150.57	7	**3**	21.3
	10	1	0.8	46.83	6	**1**	20.3
30_6_5	1	6	32.5	2541.75	11	**6**	16.5
	2	0	0.1	1.87	4	**0**	0.8
	3	17	49.6	18000.00	23	**17**	17.2
	4	7	0.5	108.94	14	**7**	15.9
	5	4	2.6	99.22	7	**4**	17.4
	6	4	0.9	1993.36	10	**4**	18.1
	7	7	15.5	10710.24	9	**7**	22.2
	8	4	0.4	23.77	8	**4**	20.3
	9	5	4.4	103.00	9	**5**	17.2
	10	2	1.2	39.21	8	**2**	17.6
30_6_6	1	3	0.1	3.98	9	4	14.3
	2	2	0.1	2.05	5	**2**	14.4
	3	13	5.0	934.72	17	**13**	14.7
	4	8	3.8	169.60	17	**8**	14.7
	5	7	0.9	10.02	9	**7**	14.8
	6	4	0.5	21.12	10	**4**	14.6
	7	7	0.8	100.86	12	**7**	15.7
	8	6	3.4	64.49	9	**6**	15.7
	9	5	0.6	10.95	12	**5**	13.6
	10	3	1.0	18.49	9	**3**	15.5

Table 5.4: Results for random instances of 30 trams stored in 6 stacks.

N_R_T	instance	OPT SM	BB sec.	CPLEX sec.	GREEDY	RTS	sec.
30_7_2	1	0	0.1	15.31	4	**0**	2.3
	2	0	4.0	13.94	1	**0**	0.7
	3	0	0.1	57.65	**0**	**0**	0.1
	4	0	0.1	126.64	**0**	**0**	0.1
	5	0	0.1	24.65	4	**0**	1.0
	6	0	0.1	78.99	10	**0**	0.2
	7	0	0.1	77.73	4	**0**	0.1
	8	0	0.1	51.59	4	**0**	11.8
	9	0	0.1	26.51	3	**0**	0.1
	10	0	0.1	11.46	**0**	**0**	0.1
30_7_3	1	1	10.6	12.08	9	**1**	27.6
	2	0	0.1	8.53	3	**0**	0.1
	3	3	4.6	1530.48	9	**3**	33.6
	4	1	22.4	$>$ 46525.00	11	**1**	31.4
	5	0	0.1	18.44	3	**0**	5.6
	6	0	1.7	9.63	14	**0**	0.6
	7	1	209.8	$>$ 43100.00	5	**1**	26.9
	8	0	0.1	3.64	5	**0**	5.1
	9	1	24331.6	88629.18	4	**1**	28.5
	10	1	0.1	2.46	3	**1**	26.4
30_7_4	1	4	4.9	344.80	19	**4**	21.6
	2	1	0.1	7.41	7	**1**	21.2
	3	5	34.5	6299.31	13	**5**	22.5
	4	2	4378.5	$>$ 40000.00	4	**2**	26.6
	5	0	0.1	4.63	7	**0**	7.0
	6	6	27.8	813.73	17	**6**	20.7
	7	6	229.8	$>$ 30000.00	8	**6**	21.5
	8	1	1.8	50.48	18	**1**	20.7
	9	3	0.2	12.76	16	**3**	19.8
	10	3	0.6	31.94	8	**3**	22.3
30_7_5	1	6	4.5	410.49	6	**6**	16.8
	2	4	34.7	893.39	4	**4**	17.0
	3	10	27.7	$>$ 7600.00	10	**10**	19.4
	4	4	142.9	20.27	4	**4**	24.6
	5	1	0.1	1.18	1	**1**	16.6
	6	11	167.9	4335.86	11	**11**	17.2
	7	5	23.8	9346.82	5	**5**	18.8
	8	1	0.1	17.30	1	**1**	18.6
	9	3	0.2	26.13	3	**3**	16.9
	10	5	1151.4	19639.08	5	**5**	23.6
30_7_6	1	7	0.6	4.64	7	**7**	14.1
	2	4	0.4	8.08	4	**4**	14.5
	3	9	8.4	20.07	9	**9**	20.8
	4	5	15.1	26.30	5	**5**	19.1
	5	5	0.6	7.82	5	**5**	13.6
	6	8	9.2	56.83	8	**8**	14.3
	7	6	13.0	259.24	6	**6**	15.7
	8	5	4.1	37.11	5	**5**	14.7
	9	4	0.9	128.29	4	**4**	14.4
	10	6	8.5	23.11	6	**6**	15.7

Table 5.5: Results for random instances of 30 trams stored in 7 stacks.

N_R_T	instance	OPT SM	BB sec.	GREEDY	RTS	sec.
40_5_2	1	0	*	11	**0**	1.7
	2	1	28.6	8	**1**	83.2
	3	2	0.1	8	**2**	89.1
	4	-	> 50000.0	12	3	84.7
	5	0	*	11	**0**	33.1
	6	3	0.1	28	**3**	88.8
	7	0	*	4	**0**	29.9
	8	0	0.1	1	1	83.3
	9	1	2.6	17	**1**	85.6
	10	0	*	6	**0**	36.4
40_5_3	1	0	*	8	**0**	0.2
	2	3	0.4	14	**3**	54.0
	3	3	0.4	14	**3**	67.2
	4	-	> 75600.0	23	15	61.4
	5	3	2473.7	13	**3**	55.4
	6	7	0.8	23	8	56.8
	7	2	1968.8	11	**2**	56.4
	8	0	*	13	**0**	8.9
	9	1	28.7	13	4	56.9
	10	4	66594.9	11	5	57.7
40_5_4	1	9	106.4	32	**9**	40.6
	2	10	399.2	20	11	42.3
	3	5	19.3	20	6	45.8
	4	14	1057.7	31	15	49.7
	5	11	187.6	24	13	45.4
	6	19	53.7	36	**19**	45.9
	7	3	5.4	8	4	42.0
	8	1	1.7	7	3	43.2
	9	2	1.4	8	**2**	42.2
	10	8	16189.9	23	**8**	43.5
40_5_5	1	14	0.1	31	**14**	32.2
	2	13	4072.4	24	**13**	33.8
	3	6	317.0	23	**6**	38.4
	4	16	220.9	30	**16**	37.8
	5	18	> 64800.0	26	18	36.6
	6	19	7.7	39	**19**	38.8
	7	11	11.6	20	**11**	35.0
	8	5	18.5	16	**5**	35.3
	9	7	40.5	9	**7**	36.3
	10	9	69.7	17	**9**	33.9
40_5_6	1	17	1.4	28	**17**	28.3
	2	13	1410.5	18	**13**	27.6
	3	13	558.9	36	**13**	34.4
	4	20	3429.8	28	**20**	30.2
	5	7	58.7	11	**7**	26.5
	6	22	86.0	33	**22**	31.5
	7	15	4.9	23	**15**	29.6
	8	20	2818.1	28	**20**	33.8
	9	11	297.5	26	**11**	29.0
	10	8	60.4	20	**8**	29.2

Table 5.6: Results for random instances of 40 trams stored in 5 stacks.

N_R_T	instance	OPT SM	BB sec.	GREEDY	RTS	sec.
40_6_2	1	1	0.1	18	**1**	84.4
	2	(2)	> 200000.0	11	2	81.5
	3	0	*	13	**0**	37.0
	4	0	*	16	**0**	64.5
	5	1	0.1	11	**1**	88.6
	6	0	*	4	**0**	7.4
	7	0	0.1	6	1	84.7
	8	0	*	4	**0**	1.7
	9	0	*	8	**0**	0.3
	10	0	*	3	**0**	0.2
40_6_3	1	1	0.1	13	**1**	58.4
	2	1	4655.8	7	**1**	53.2
	3	1	0.1	10	**1**	56.3
	4	0	*	11	**0**	13.0
	5	1	0.1	7	**1**	61.1
	6	8	2130.3	26	10	60.2
	7	1	40484.3	5	**1**	58.8
	8	1	605.8	22	2	59.2
	9	1	0.1	9	**1**	51.7
	10	0	*	12	**0**	4.6
40_6_4	1	3	9.4	27	**3**	45.6
	2	7	213.6	12	**7**	42.4
	3	4	1259.1	8	5	44.8
	4	1	0.1	10	**1**	49.7
	5	5	30.6	15	6	46.7
	6	(13)	> 86400.0	24	13	48.6
	7	2	6.2	12	**2**	44.2
	8	(3)	> 79200.0	8	3	54.3
	9	1	0.1	9	**1**	42.9
	10	2	33.5	16	**2**	48.8
40_6_5	1	2	0.1	14	**2**	34.9
	2	8	744.1	18	9	34.7
	3	4	23.2	21	**4**	35.2
	4	0	*	13	**0**	2.4
	5	8	897.9	11	**8**	40.7
	6	(16)	> 82800.0	30	16	40.7
	7	9	170.7	34	**9**	36.6
	8	12	1126.6	39	**12**	38.3
	9	4	32314.1	14	5	34.0
	10	1	5.2	13	**1**	34.9
40_6_6	1	4	8.6	16	**4**	32.1
	2	14	644.9	21	**14**	29.7
	3	10	142.9	30	**10**	29.2
	4	1	0.1	13	**1**	30.1
	5	6	1.0	10	**6**	32.4
	6	21	5034.4	35	**21**	34.7
	7	16	1171.1	41	**16**	31.8
	8	6	260.8	19	7	33.7
	9	5	10.4	22	**5**	26.7
	10	4	5.0	17	4	32.0

Table 5.7: Results for random instances of 40 trams stored in 6 stacks.

N_R_T	instance	OPT SM	BB sec.	GREEDY	RTS	sec.
50_5_2	1	0	0.1	16	**1**	149.1
	2	(3)	> 129600.0	18	3	150.1
	3	0	*	23	**0**	40.4
	4	0	*	3	**0**	55.4
	5	0	*	17	**0**	110.3
	6	2	0.1	8	**2**	150.2
	7	0	*	12	**0**	8.5
	8	(5)	> 100000.0	13	5	158.3
	9	0	0.1	12	3	151.3
	10	1	0.3	19	2	148.5
50_5_3	1	(12)	> 75600.0	39	12	105.1

Table 5.8: Results for random instances of 50 trams stored in 5 stacks.

N_R_T	instance	GREEDY	RTS	sec.	N_R_T	instance	GREEDY	RTS	sec.
50_5_3	1	39	12	105.1	50_5_5	1	36	14	62.4
	2	36	16	103.7		2	31	18	67.8
	3	19	13	110.6		3	19	12	70.8
	4	38	12	106.5		4	40	21	64.0
	5	40	15	114.2		5	46	19	66.3
	6	18	5	99.4		6	20	7	62.4
	7	17	6	100.5		7	35	8	63.2
	8	33	5	106.4		8	29	17	67.6
	9	21	5	101.8		9	56	28	66.5
	10	16	10	103.0		10	25	18	66.9
50_5_4	1	28	13	77.2	50_5_6	1	46	26	53.5
	2	27	12	77.6		2	58	39	56.9
	3	19	10	80.5		3	45	23	56.9
	4	31	18	78.9		4	35	25	53.3
	5	45	17	82.7		5	37	31	57.9
	6	27	14	80.9		6	35	14	53.0
	7	30	6	78.8		7	44	21	55.3
	8	40	17	87.7		8	41	22	59.0
	9	24	8	79.2		9	34	25	57.1
	10	32	17	80.7		10	37	24	57.5

Table 5.9: Results for GREEDY-DTDP and RTS for random instances of 50 trams stored in 5 stacks.

5.6 Minimizing Type Mismatch at Departure

In Section 3.5 and Section 5.2, we presented a dynamic programming approach for DTMP. For fixed number of stacks, this dynamic programming approach yields an optimal solution for DTMP in polynomial time. We implemented this approach following the description in Section 3.5 and Section 5.2. We call the corresponding algorithm **TYPE**. Beginning with the first departure, TYPE proceeds with searching for an optimal solution by breadth-first-search. TYPE is applied to several random instances introduced in the previous section. For each setting of N_R_T, we generate 100 random instances of this kind. In Table 5.10 and Table 5.11, we present the corresponding computational results. We observe that all the instances can be solved with an average computation of less than 400 seconds on a Hewlett Packard HP-9000-735 workstation with 144 MByte core memory.

T	$N = 25$, $R = 5$		$N = 30$, $R = 5$		$N = 40$, $R = 5$		$N = 50$, $R = 5$	
	TM	CPU [s]	TM	CPU [s]	TM	CPU [s]	TM	CPU [s]
2	0.4	0.64	0.0	2.69	1.4	27.02	0.6	175.23
3	1.9	0.61	2.1	2.66	2.1	26.70	2.1	176.73
4	3.1	0.63	2.7	2.70	3.5	26.97	5.6	175.92
5	4.4	0.60	4.5	2.78	6.0	27.15	7.6	174.54
6	5.1	0.66	5.3	2.71	6.6	26.83	9.1	174.84

Table 5.10: Results for TYPE applied to random instances of 25 to 50 trams of 2 to 6 types stored in 5 stacks. Average number of type mismatches and average computation times for 100 instances of size N_R_T.

T	$N = 30$, $R = 6$		$N = 30$, $R = 7$		$N = 40$, $R = 6$	
	TM	CPU [s]	TM	CPU [s]	TM	CPU [s]
2	0.0	22.65	0.4	135.94	0.6	392.21
3	1.4	22.63	1.0	135.88	1.9	392.55
4	3.0	22.62	1.8	135.94	3.2	392.44
5	3.6	22.32	3.1	136.13	4.5	392.11
6	4.1	22.61	4.2	136.62	6.1	392.91

Table 5.11: Results for TYPE applied to random instances of 30 trams of 2 to 6 types stored in 6 and 7 stacks and for 40 trams of 2 to 6 types stored in 6 stacks. Average number of type mismatches and average computation times for 100 instances of size N_R_T.

5.6.1 An Approximation Algorithm for DTDP

In Section 4.3, we investigated the construction of a feasible solution for TMP given a feasible solution for TDP and vice versa. Theorem 4.3.14 implies that we can obtain a feasible solution for TDP from a given optimal solution for TMP. By Theorem 4.1.7, it is possible to extend this observation of DTDP and DTMP. A feasible solution for DTDP does not require more than $t \cdot max_r P_r$ shunting movements where t denotes the optimal value for the corresponding DTMP and P_r denotes the length of stack $\mathcal{P}_r$, $1 \leq r \leq R$.

For fixed R, Theorem 3.5.34 implies a polynomial-time dynamic programming approach for DTMP. Consequently, in polynomial-time we obtain a feasible solution for DTDP requiring at most $t \cdot max_r P_r$ shunting movements.

However, the results obtained by this approximation algorithm are worse than the results that we achieve by applying the RTS heuristic (cf. Table 5.3 – Table 5.11).

5.7 Conclusion

The tram dispatch problem at departure (DTDP) occurs when trams already standing in the depot have to be dispatched to the departures corresponding to the round trip of the next schedule period. This problem may also be of interest for TDP. On arrival, the dispatcher may decide to assign the trams to locations in the depot without regarding the next departures. In this case, it is always possible to assign the trams to the stack position without shunting. After the last tram has arrived, the dispatcher has time to determine an assignment of trams to departures.

Using the exact methods introduced above, we can determine an optimal solution for instances of 30 to 40 trams in less than one hour computation time. Moreover, the RTS heuristic already yields near optimal upper bounds. Consequently, the algorithms introduced in this chapter can be applied to real-world problem and integrated in decision support systems for tram dispatch.

Since RTS and the dynamic programming approach TYPE for DTMP require only computation times of only a few minutes, both algorithm can be applied to the following situation, too.

When a tram breaks down before its departure, the trams standing behind this tram in the same stack cannot leave the depot. However, the corresponding round trips which are assigned to these trams have to be served with a minimum delay. An updated assignment of trams to departures is required within a short time interval. For instances of 30 to 50 trams, such an assignment can be computed by RTS and TYPE within the required time of two to five minutes. For larger instances, both algorithms can be used to compute a partial assignment which assigns trams to the next departures. When the other trams become accessible

again, the assignment for the remaining departures can be updated.

In conclusion, the algorithms introduced in this chapter are applicable to real-world instances of several depots. Moreover, they can be applied to situation where real-time effects require an updated assignment within short time intervals.

Chapter 6

Online Problems

Before we start with our investigations in the online tram dispatch problem, we review some methods for the analysis of online problems and recent results. We give a brief introduction to competitive analysis. Then, we present and discuss some "benchmark" online problems arising in computer science. We conclude with an overview of online versions of standard combinatorial optimization.

6.1 Competitive Analysis

Starting with the work of Sleator and Tarjan [ST85] on the list update problem and the paging problem, the study of online problems and online algorithms became more and more of interest within the last decade. Sleator and Tarjan examined and compared the **amortized complexity** [Tar85] of several list search and paging heuristics. The resulting approach – later denoted **competitive analysis** [KMRS88] – is a variation of worst-case analysis for online algorithms. In the 1960s, this approach was used implicitly in the early works on bin packing [Gra66]. Similar techniques have found application to financial problems in the sixties and seventies [FW98].

Definition 6.1.1: An **online problem** Π is a (minimization) problem where the complete instance is not known in advance but revealed sequentially. Before the next part of the instance is given, a solution for the partial instance, i.e., for the part of the instance known so far, has to be generated.

An **online algorithm** is an algorithm which solves the online problem Π, i.e., produces a sequence of correct solutions for the sequence of partial instances.

An online algorithm is said to be **c-competitive** if and only if the following inequality holds for every problem instance I

$$cost_{\mathrm{ALG}}(I) \leq c \cdot cost_{\mathrm{OPT}}(I) + b$$

where $cost_{\mathrm{ALG}}(I)$ denotes the cost of the solution produced by ALG for the instance I, $cost_{\mathrm{OPT}}(I)$ is the optimal cost computed by an optimal algorithm

knowing the complete instance I in advance, and b is a constant independent of I. OPT is called **optimal offline algorithm**.

The infimum over all factors c for which ALG is c-competitive is called **competitive ratio** of ALG. The infimum of all competitive ratios of algorithms solving Π is called **competitive ratio** of Π. An algorithm is said to be **strongly competitive** if it achieves this best possible competitive ratio. ❑

In the following, we restrict ourselves to minimization problems. For maximization problems, the concept of competitive analysis can be defined in an analogous way (cf. [FW98]).

6.1.1 The Paging Problem

One well-studied online problem is the **paging problem** [Ira98]. In this problem, one has to manage the access to a two-level memory system consisting of a small but fast memory (cache) and a large slow memory. The complete memory is divided into pages of uniform size.

A page can only be accessed when it is stored in the cache. For a given sequence of page requests, the goal is to determine which pages should be removed from the cache in order to transfer the pages to be accessed from the slow memory into the cache. The number of such **page faults** shall be minimized.

In the case where the complete request sequence is known in advance, an optimal strategy for the paging problem works as follows [Bel66]: If the requested page is not in the cache, evict the page whose next request occurs furthest in the future.

In the online paging problem, we have to decide at each request which page to evict in case of a page fault without knowing the request sequence. Sleator and Tarjan [ST85] examine two online algorithms – **Least-Recently-Used (LRU)** and **First-In-First-Out (FIFO)**, defined in the following way:

- LRU: On a page fault, evict the page that was accessed least recently
- FIFO: On a page fault, evict the page that was transferred into the cache least recently

Sleator and Tarjan show that both algorithms are k-competitive where k denotes the number of pages that can be stored in the cache. Moreover, both algorithms are shown to be strongly competitive.

In order to prove the lower bound for the competitive ratio of any arbitrary online algorithm for the paging problem, we construct an instance with a request sequence of exactly $k + 1$ pages: the k pages stored in the cache and one page stored in the slow memory. By ALG, we denote the arbitrarily chosen online paging algorithm. We always request the page which is stored outside the cache so that ALG fails on every request. We assume that ALG and an optimal offline

algorithm OPT start with the same set of pages stored in the cache. Otherwise, the first page request causes a page fault only for ALG. The optimal offline algorithm OPT by Belady [Bel66] knows the complete request sequence. OPT always evicts the page whose request occurs furthest in the future. The next page fault for OPT occurs when the page removed from the cache is requested again. Since this request occurs furthest in the future, at least $k-1$ pages will be requested between two faults. Hence, OPT fails on at most every k-th page whereas ALG fails on every page. This yields the lower bound of k on the competitive ratio of an arbitrary online algorithm.

The construction of the above request sequence can be regarded as being given by a **cruel adversary**. We will consider this proof technique more intensively in Section 6.1.2 and in Section 6.2. In Chapter 7, we will apply this technique in order to proof lower bounds on the competitive ratio for the tram dispatch problems.

Karlin et al. [KMRS88] examine a class of paging algorithms including LRU. The algorithms of this class are called **marking algorithms**. A marking algorithm proceeds in **phases**. Initially, all pages are unmarked. Whenever a page is requested, it is marked. On a page fault, an unmarked page is evicted. After every page is marked, a phase ends just before the next fault occurs. Then, all pages are unmarked and a new phase starts. Karlin et al. [KMRS88] show that every marking algorithm is k-competitive.

The result can easily be achieved if we examine the performance of an arbitrary marking algorithm ALG and an optimal offline algorithm OPT for one phase. OPT must fail at least one time per phase (possibly except of phase one). After the first request of a phase, OPT has the requested page in the cache. If OPT does not fail during the rest of the phase, then it must have in the cache all k pages requested during the phase. The first request of the next phase is, by definition, not in this set. Hence, OPT cannot avoid a page fault. ALG marks every page whenever it is requested. These pages will not be evicted during the remainder of the phase. Since there are exactly k distinct pages requested in one phase, any marking algorithm produces at most k page faults in each phase. Consequently, ALG is k-competitive.

In some sense, a marking algorithm uses the recent requests to predict the next requests. Every marked page has been requested more recently than any of the unmarked pages. A marking algorithm assumes that the pages that have been recently requested are more likely to be requested again. The optimal offline algorithm introduced by Belady has low cost on these sequences that exhibit this locality of reference. This motivates the good performance of marking algorithms.

As mentioned before, competitive analysis can be regarded as a worst-case measure for the performance of online algorithms. From the theoretical point of view, LRU and FIFO are equivalent in the sense that they are k-competitive [BEY98]. From a practical point of view, LRU outperforms FIFO [Ira98].

Chrobak and Noga [CN99] recently proved a conjecture by Borodin et al.

[BIRS95] for a modified competitive ratio. Borodin et al. introduce an access graph G whose vertices are the pages. Two vertices are connected by an edge if the two corresponding pages can be requested directly after the other. Each feasible sequence of requests corresponds to a walk in G. If we restrict ourselves on such instances I defined by G, LRU achieves a competitive ratio not greater than the ratio of FIFO [Ira98].

6.1.2 Randomization

According to the work of Ben-David et al. [BDBK+94], we present the competitive analysis in a more general framework. The input is presented iteratively to the online algorithm which has to compute the input straightly. Hence, we study the performance of those algorithms in the framework of request-answer-games.

A **request-answer-game** is a two-person-game in which the input (or request) sequence is determined by a player called **adversary**. The second player is an online algorithm that has to serve each request before the adversary presents its next request. More formally, we define a request-answer-game as follows.

Definition 6.1.2: A request-answer-game consists of a request set R, a finite answer set A, and cost functions $f_n : R^n \times A^n \to \mathbb{R}_+ \cup \{\infty\}$, $n \in \mathbb{N}_0$. ❑

The cost caused by the answers $a_1, \ldots, a_n$ on the requests $r_1, \ldots, r_n$ are represented by the cost functions f_n. Here, $f_n\big((r_1, \ldots, r_n), (a_1, \ldots, a_n)\big)$ are the cumulative costs caused by the online algorithm while iteratively answering the requests up to r_n.

In the following, we distinguish between **deterministic** and **randomized** online algorithms.

Definition 6.1.3: A **deterministic online algorithm** ALG is a sequence of functions $g_i : R^i \to A$ for $i \in \mathbb{N}$. For any sequence request $r = (r_1, r_2, \ldots, r_n) \in R^n$, we define $G(r) := (a_1, a_2, \ldots, a_n) \in A^n$ with $a_i := g_i(r_1, \ldots, r_i)$ for $i = 1, \ldots, n$. The **cost of an online algorithm** ALG on sequence r is $cost_{\mathrm{ALG}}(r) = f_n(r, G(r))$. The **optimal cost** for the same request sequence r is defined by $cost^*(r) = \min\{f_n(r, a) \mid a \in A^n\}$.

A **randomized online algorithm** RAND is a probability distribution over all deterministic online algorithms ALG. For any request sequence r the answer sequence $G(r)$ as well as the corresponding cost $cost_{\mathrm{RAND}}$ are random variables. ❑

The request sequence $r = (r_1, \ldots, r_n)$ is generated by the adversary. The adversary chooses the length of the request sequence. We distinguish three kinds of adversaries by their strategies of choosing the request sequence and the way in which they determine their (optimal) sequence of answers.

Definition 6.1.4: An adversary is called

- **oblivious adversary** if it constructs the request sequence independently to the answers of the online algorithm, i.e., an oblivious adversary may be regarded as a player who chooses its strategy in advance ignoring the actions made by its opponent.
- **adaptive online adversary** if it constructs the next request r_i depending on the answers $a_1, \ldots, a_{i-1}$ of the online algorithm. However, the adaptive online adversary must serve the same request before the next answer of the online algorithm is known. Therefore, this adversary itself can be regarded as an online algorithm.
- **adaptive offline adversary** if the next request constructed by this adversary depends on the answers $a_1, \ldots, a_{i-1}$ of the online algorithm. In contrast to the adaptive online adversary, the adaptive offline adversary serves the complete request sequence optimally after the last request is generated.

❑

We are now able to define a measure for the performance of online algorithms with respect to the opposing adversary. First, we compare the cost of an online algorithm to the optimal cost of an oblivious adversary.

Definition 6.1.5: A deterministic online algorithm is called **c-competitive against any oblivious adversary**, $c \in \mathbb{R}_+$, if there is a fixed $b \in \mathbb{R}_+$ so that for all $n \in \mathbb{N}$ and all $r \in R^n$

$$cost_{\mathrm{ALG}}(r) \leq c \cdot cost^*(r) + b \tag{6.1.1}$$

A randomized online algorithm RAND is called **c-competitive against any oblivious adversary**, $c \in \mathbb{R}_+$, if for all request sequences r

$$E[cost_{\mathrm{RAND}}(r)] \leq c \cdot cost^*(r) + b \tag{6.1.2}$$

The infimum of all c so that ALG (RAND) is c-competitive is called **competitive ratio** of ALG (RAND) and is denoted by $\mathcal{C}^{obl}_{\mathrm{ALG}}$ (and $\mathcal{C}^{obl}_{\mathrm{RAND}}$, respectively).
❑

By Definition 6.1.4, adaptive adversaries react on the answer made by the online algorithm. Therefore, the derived cost depends on the interaction between the online algorithm and the adaptive adversary.

An adaptive offline adversary ADOFF can be regarded as a sequence of functions $q_i : A^i \to R \cup \{\mathrm{stop}\}$, where $i = 0, 1, \ldots, d_{\mathrm{ADOFF}}$ and the last function $q_{d_{\mathrm{ADOFF}}} = \mathrm{stop}$ corresponds to the "end-of-game" message in the request-answer-game. Given a deterministic online algorithm ALG, we can define for

ADOFF the actual request and answer sequences $r = r(\mathrm{ALG}, \mathrm{ADOFF})$ and $a = a(\mathrm{ALG}, \mathrm{ADOFF})$. The length $n = d_{\mathrm{ADOFF}}$ of both sequences is also dependent on ALG and ADOFF. We define recursively $r_{i+1} = q_i(a_1, \ldots, a_i)$ for $i = 0, \ldots, n-1$. The answer sequence is determined by ALG in the same way as described above, i.e., $a = G(r)$. The cost of the online algorithm ALG against the adaptive adversary ADOFF are defined analogously to Definition 6.1.3 as $cost_{\mathrm{ALG,ADOFF}}(r) = f_n(r, a)$. The cost of the adaptive offline adversary are defined as $cost_{\mathrm{ADOFF,ALG}}(r) = cost^*(r)$. Note that, r as well as a depends on ALG and ADOFF.

For an adaptive online adversary ADON, we have to introduce another answer sequence $\alpha = (\alpha_1, \ldots, \alpha_n)$ because ADON must serve its own request sequence simultaneously to ALG. The sequence b is determined by $\alpha_{i+1} := p_i(a_1, \ldots, a_i)$ where $p_i : A^i \to A$ describes the reaction of ADON on its own request r_i which depends on the answers of ALG. Hence, the cost of an adaptive online adversary are defined by $cost_{\mathrm{ADON,ALG}} = f_n(r, \alpha)$.

We can repeat the above construction for randomized online algorithms. In this case, we have to deal with random variables.

Analogously to the definition of competitiveness against any oblivious adversary, we give the following definition:

Definition 6.1.6: A deterministic online algorithms ALG is called **c-competitive against any adaptive online adversary** ADON, $c \in \mathbb{R}_+$, if there is a fixed $b \in \mathbb{R}$ so that for all $n \in \mathbb{N}$ and all request sequences $r \in R^n$

$$cost_{\mathrm{ALG,ADON}}(r) \leq c \cdot cost_{\mathrm{ADON,ALG}}(r) + b \tag{6.1.3}$$

and **c-competitive against any adaptive offline adversary** ADOFF, $c \in \mathbb{R}_+$, if

$$cost_{\mathrm{ALG,ADOFF}}(r) \leq c \cdot cost_{\mathrm{ADOFF,ALG}}(r) + b \tag{6.1.4}$$

A randomized online algorithms RAND is called **c-competitive against any adaptive online adversary** ADON, $c \in \mathbb{R}_+$, if there is a fixed $b \in \mathbb{R}$ so that for all $n \in \mathbb{N}$ and all request sequences $r \in R^n$

$$E[cost_{\mathrm{RAND,ADON}}(r)] \leq c\, E[cost_{\mathrm{ADON,RAND}}(r)] + b \tag{6.1.5}$$

and **c-competitive against any adaptive offline adversary** ADOFF, $c \in \mathbb{R}_+$, if there is a fixed $b \in \mathbb{R}$ so that for all $n \in \mathbb{N}$ and all request sequences $r \in R^n$

$$E[cost_{\mathrm{RAND,ADOFF}}(r)] \leq c\, E[cost_{\mathrm{ADOFF,RAND}}(r)] + b \tag{6.1.6}$$

❑

An adaptive offline adversary is a strong opponent against an online algorithm. It may react on each decision made by the online algorithm and it is able to wait with serving the input sequence until the last input is generated.

The best possible competitive ratios against the different adversaries can be related in the following way:

$$\inf_{\text{RAND}} c^{obl}_{\text{RAND}} \leq \inf_{\text{RAND}} c^{\text{ADON}}_{\text{RAND}} \leq \inf_{\text{RAND}} c^{\text{ADOFF}}_{\text{RAND}} \leq \inf_{\text{det. ALG}} c.$$

The next theorem by Ben-David et al. [BDBK+94] shows that randomization does not help against the adaptive offline adversary.

Theorem 6.1.7: If RAND is a randomized online algorithm that is c-competitive against any adaptive offline adversary, then there is a deterministic online algorithm ALG that is c-competitive against any adaptive offline adversary.

Proof: We consider the following request-answer-game between the deterministic online algorithm ALG and an arbitrary adaptive offline adversary ADOFF. In every step of the game, ADOFF gives a request r_i to ALG that ALG has to serve it before the next request is generated. We call each pair $p_i = (r^i, a^i)$ with $r^i := (r_1, \ldots, r_i)$ and $a^i := (a_1, \ldots, a_i)$ a **position** in the game. A position p_i is said to be **immediately winning** for ADOFF if $f_i(r^i, a^i) > c \cdot cost^*(r^i)$. A position p_i is called **winning** if there is an adaptive rule for generating requests so that an immediately winning position $p_{i'}$ is reachable from p_i within t steps, $t \in \mathbb{N}$, regardless of how ALG reacts on the requests. For instance, the initial position p^0 is winning for ADOFF if and only if for any deterministic algorithm ALG there is a constant $b \in RR_+$ and an index $t \in \mathbb{N}$ so that

$$cost_{\text{ALG,ADOFF}}(r^t) > c\, cost_{\text{ADOFF,ALG}}(r^t) + b.$$

A player is said to have a **winning strategy** if the initial position is winning for this player.

First, we assume that ADOFF has a winning strategy against any deterministic online algorithm ALG. This implies that

$$cost_{\text{ALG,ADOFF}}(r^t) > c\, cost_{\text{ADOFF,ALG}}(r^t) + b.$$

for some $t \in \mathbb{N}$ and $b \in \mathbb{R}_+$. A randomized online algorithm RAND is a probability distribution over all deterministic online algorithms. Hence,

$$E[cost_{\text{RAND,ADOFF}}(r^t)] > c\, E[cost_{\text{ADOFF,RAND}}(r^t)] + b.$$

Consequently, no randomized online algorithm can be c-competitive against ADOFF. This contradicts the assumption of Theorem 6.1.7. We achieve that ADOFF does not have a winning strategy against any deterministic online algorithm.

Secondly, to complete the proof we show that there is a deterministic online algorithm ALG that has a winning strategy against the (arbitrary) adaptive offline adversary ADOFF.

A position $p_i = (r^i, a^i)$ is a winning position for ADOFF if and only if there is a request r_{i+1} so that for every answer a_{i+1} the next position p_{i+1} is again winning for ADOFF. If p_i is not winning for ADOFF, ALG can answer every new request r_{i+1} by an answer a_{i+1} so that p_{i+1} is not winning for ADOFF. Since every position p_i is not a winning position for ADOFF, ALG is able to avoid that ADOFF reaches an immediately winning position. Such a strategy is winning for ALG which implies that ALG is c-competitive against the arbitrary offline adversary ADOFF. ❑

Ben-David et al. [BDBK$^+$94] relate the power of the three adversaries introduced in Definition 6.1.4. They proved the following results (see Theorem 6.1.8). We denote by ALG a deterministic online algorithm, by RAND, RAND_1, and RAND_2 randomized online algorithms, by ADON arbitrary online adaptive adversaries, and by ADOFF arbitrary adaptive offline algorithms.

Theorem 6.1.8:

1. Suppose that RAND_1 is c-competitive against ADON and RAND_2 is d-competitive against any oblivious adversary. Then RAND_2 is $c \cdot d$-competitive against ADOFF.

2. If RAND_1 is c-competitive against ADON and RAND_2 d-competitive against any oblivious adversary, then there is a deterministic online algorithm which is $c \cdot d$-competitive.

3. If RAND is c-competitive against ADON, then RAND is c^2-competitive against ADOFF which implies that there is a c^2-competitive deterministic online algorithm.

Proof: see Ben-David et al. [BDBK$^+$94] ❑

Theorem 6.1.7 and Theorem 6.1.8 show that if there is a good competitive randomized algorithm, then there exists also a good deterministic online algorithm. Unfortunately, the related proofs are not constructive. For given randomized algorithms, the question of constructing deterministic online algorithms has been investigated by Deng and Mahajan [DM97]. Deng and Mahajan proved that there is a request-answer-game for which there is a 1-competitive (computable) randomized online strategy but no (computable) deterministic one being at least c-competitive for any $c > 0$. The reader may consult [BDBK$^+$94] and [DM97] for details.

6.1.3 Back to the Paging Problem

Next, we survey some results for randomized algorithms for the online paging problem. Raghavan and Snir [RS89] consider algorithms which do not use memory space during computation. LRU and FIFO need memory for storing and determining the page to be evicted as the next. Raghavan and Snir show that any memoryless online algorithm has a competitive ratio of at least k against an oblivious adversary.

The randomized version of the marking algorithm has a competitive ratio $2H_k$ against any oblivious adversary [FKL+91], where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k-th harmonic number.

We denote by c_{ALG}^P the competitive ratio that ALG achieves for instances corresponding to P if such a competitive ratio exists. For c_{ALG}^P, a lower bound of H_k can be derived by defining a probability distribution P for which c_{ALG}^P is at least H_k for all deterministic algorithms ALG [FKL+91]. We define S to be the set of $k+1$ pages including the k pages stored in the cache. Then, P is defined to be the uniform distribution over S. Let n denote the length of the instance's request sequence. Then,

$$E_P[cost_{\mathrm{ALG}}(I)] = \frac{n}{k+1}$$

Once again, we divide the sequence into r phases (cf. page 123). The optimal offline algorithm OPT serves the request with at most $r+1$ faults. The number of phases is a random variable. Hence,

$$E_P[cost_{\mathrm{OPT}}(I)] \leq E_P[r+1]$$

The durations of the phases are independent, identically distributed random variables, denoted by X_i. We observe that

$$\lim_{n\to\infty} \frac{n}{E_P[r]} = E_P[X_i]$$

where $E_P[X_i]$ is the expected length of phase i. With reference to the coupon collection problem [MR95], we obtain that $E_P[X_i] = (k+1)H_k$. This leads to

$$\lim_{n\to\infty} \frac{E_P[cost_{\mathrm{ALG}}(I)]}{E_P[cost_{\mathrm{OPT}}(I)]} \geq \frac{E_P[X_i]}{k+1} = H_k.$$

6.2 Lower and Upper Bounds

For the paging problem, we introduced some online algorithms which have been proven to be k-competitive. Usually, the competitiveness of an online algorithm is shown in two steps. First, a lower bound on the competitive ratio is derived.

This lower bound is typically a general lower bound for all online algorithms. Secondly, an upper bound is determined for the specific online algorithm.

For the deterministic case, we examine a certain instance for which any online algorithm behaves poorly. This standard technique can be regarded in a game-theoretic sense. A – so called – **cruel adversary** constructs an instance for which the considered online algorithm needs high cost whereas an optimal offline algorithm is able to produce a low cost solution.

For randomized algorithms, a lower bound can be determined using **Yao's Min-Max-Principle** [Yao77]. We prove a lower bound for randomized online algorithms in the following way. First, we introduce a specified probability distribution P. Then, we restrict ourselves on instances that are generated according to P. More formally, we introduce the following notation.

Definition 6.2.9: Let ALG be a deterministic online algorithm and P be a given probability distribution. ALG is said to be **c-competitive under P** if

$$E_P[cost_{\mathrm{ALG}}(I)] \leq c \cdot E_P[cost_{\mathrm{OPT}}(I)] + b, \quad b \in \mathbb{R},$$

for all instances I that are generated according to P.

Let $\mathcal{C}^P_{\mathrm{ALG}}$ denote the infimum over the competitive ratios achieved by ALG. ❑

Yao's Min-Max-Theorem [Yao77] states that the competitive ratio of the best possible randomized online algorithm against any oblivious adversary is equal to the competitive ratio of the best deterministic algorithm.

$$\inf_{\mathrm{RAND}} \mathcal{C}^{obl}_{\mathrm{RAND}} = \sup_{P} \inf_{\mathrm{ALG}} \mathcal{C}^P_{\mathrm{ALG}} \tag{6.2.7}$$

Consequently, a lower bound for $\mathcal{C}^P_{\mathrm{ALG}}$ is a lower bound for $\mathcal{C}^{obl}_{\mathrm{RAND}}$. This technique is used in the proof that the randomized marking algorithm is $2H_k$-competitive [FKL+91].

Relating the Adversaries

We define $\mathcal{C}^{obl}_{\mathrm{RAND}}$ to be the competitive ratio of RAND against any arbitrary oblivious adversary. Analogously, we denote by $\mathcal{C}^{\mathrm{ADON}}_{\mathrm{RAND}}$ and $\mathcal{C}^{\mathrm{ADOFF}}_{\mathrm{RAND}}$ the competitive ratio of RAND against any adaptive online and any adaptive offline adversary. Then, the following inequality holds:

$$\mathcal{C}^{obl}_{\mathrm{RAND}} \leq \mathcal{C}^{\mathrm{ADON}}_{\mathrm{RAND}} \leq \mathcal{C}^{\mathrm{ADOFF}}_{\mathrm{RAND}}$$

If $\mathcal{C}^{obl}$, $\mathcal{C}^{\mathrm{ADON}}$, and $\mathcal{C}^{ADOFF}$ denote the problem's competitive ratios against arbitrary oblivious, adaptive online, and adaptive offline adversaries, we can relate

these values to the best competitive ratio $\mathcal{C}^{det}$ that can be achieved by a deterministic online algorithm:

$$\mathcal{C}^{obl} \leq \mathcal{C}^{\text{ADON}} \leq \mathcal{C}^{ADOFF} \leq \mathcal{C}^{det}$$

Hence, we obtain lower bounds on the competitive ratios of deterministic online algorithms by considering the performance of randomized online algorithms.

Potential Functions

The construction of a **potential function** is a common technique for proving upper bounds on the competitive ratio of a given online algorithm ALG. A potential function Φ is a non-negative real-valued mapping of the configurations produced by ALG and OPT. Let I be the underlying instance of the online problem where I consists of m requests. Then, $\Phi(i)$ denotes the value of the potential function after both ALG and OPT have computed the first i requests. We denote by $cost^{(i)}_{\text{ALG}}$ the cost of ALG for serving the i-th request, and by $cost^{(i)}_{\text{OPT}}$ the cost of OPT on the same request. We define the **amortized cost** a_i as follows

$$a_i := cost^{(i)}_{\text{ALG}} + \Phi(i) - \Phi(i-1)$$

We use this definition for the analysis of the complete sequence of requests. An upper bound for the competitive ratio of ALG can be derived in the following way. If we assume that a_i is bounded from above by $c \cdot cost^{(i)}_{\text{OPT}}$ for all i, $c \geq 0$, then we obtain that

$$\begin{aligned}
cost_{\text{ALG}}(I) - \Phi(0) &= \sum_{i=1}^{m} cost^{(i)}_{\text{ALG}} - \Phi(0) \\
&\leq \Phi(m) - \Phi(0) + \sum_{i=1}^{m} cost^{(i)}_{\text{ALG}} \\
&= \sum_{i=1}^{m} \left(cost^{(i)}_{\text{ALG}} + \Phi(i) - \Phi(i-1) \right) \\
&= \sum_{i=1}^{m} a_i \\
&\leq \sum_{i=1}^{m} c \cdot cost^{(i)}_{\text{OPT}} = c \cdot cost_{\text{OPT}}(I) \\
\Rightarrow cost_{\text{ALG}}(I) &\leq c \cdot cost_{\text{OPT}}(I) + \Phi(0)
\end{aligned}$$

Such an analysis of a sequence of requests is called **amortized analysis** [Tar85]. This notation is motivated by the way how the costs of ALG and OPT are

compared. Each individual cost on a single request may be costly for ALG compared to the cost incurred by OPT. However, the cost are in some sense "averaged" over the instance's complete request sequence.

The difficulty in proving upper bounds lies in the right choice of a potential function. A typical potential function is constructed in such a way that it measures the "distance" or "difference" between the configuration of ALG and the configuration of OPT.

6.3 The k-Server Problem and Metrical Tasks Systems

Another well-studied online problem is the **k-server problem** which has been introduced by Manasse, McGeoch, and Sleator [MMS88]. In this problem, k mobile servers have to serve a sequence of requests. Initially, each server is located at a given point in an underlying metric space $\mathcal{M}$. A requests consists of a point p in $\mathcal{M}$. If a server is at p, then nothing has to be done. Otherwise, a server has to be moved from its position to p. The cost for this operation corresponds to the distance in $\mathcal{M}$ between the server's former position and p. The objective is to minimize the total cost, i.e., the total distance needed to serve all the requests. The only information known to an online algorithm are the initial server positions and the metric space $\mathcal{M}$. An optimal offline strategy for the k-server problem is due to [CKPV91] and can be computed in $O(kn^2)$.

The paging problem is a special case of the k-server problem. In this case, $\mathcal{M}$ is the uniform metric space where the distance between any two distinct points is equal to one. The cost for remaining at a fixed point is zero. Consequently, we obtain a lower bound of k for the competitive ratio of the k-server problem.

Koutsoupias and Papadimitriou [KP94] show that the **work function algorithm** is $(2k-1)$-competitive. The work function algorithm WFA uses the following idea. WFA computes the server configuration that minimizes the sum of the retrospective offline cost for the request sequence known so far and the greedy distance needed to obtain this configuration from the current one. For $k = 2$ and $k = n + 1$, Manasse et al. [MMS88] proved the upper bound of k for the competitive ratio. It remains an open question whether or not there is a k-competitive algorithm for the general case.

A **Metrical task system**, introduced by Borodin, Linial, and Saks [BLS92], forms a general model for describing a class of (online) problems. A **task system** consists of a finite set of states S, a $(\mid S \mid \times \mid S \mid)$-dimensional matrix $D = (d_{ij})$ of distances between every two states, and a set T of tasks to be served. The distances d_{ij} satisfy the triangle inequality. A **task** $t \in T$ is a non-negative $\mid S \mid$-dimensional vector whose i-th component t_i denotes the cost of processing task t in state $i \in S$. A task system is called **metrical task system** if D is symmetric.

An online algorithm for a metrical task system has to produce a **schedule** $\pi : \{1, \ldots, n\} \to S$ for the input sequence of tasks $\sigma = (t^1, t^2, \ldots, t^n)$. Here, $\pi(i) = j$ denotes that task t^i is processed in state j. The **cost of schedule** π **on** σ is the sum of the cost of moving from state to state and the cost of processing the tasks, i.e.,

$$cost(\pi, \sigma) := \sum_{i=1}^{n} d_{\pi(i-1),\pi(i)} + \sum_{i=1}^{n} t^i_{\pi(i)} \tag{6.3.8}$$

By $\pi(0)$, we denote the initial state which is fixed for each algorithm. The schedule obtained by the online algorithm ALG is denoted by $\mathrm{ALG}(\sigma)$. The cost of ALG is $cost(\mathrm{ALG}(\sigma), \sigma)$. The cost of the optimal oblivious adversary ADV is equal to the cost of the optimal offline algorithms and is defined in accordance with the previous section to be $cost^*(\sigma) = \min_\pi cost(\pi, \sigma)$.

A lower bound for the competitive ratio of online algorithms for metrical task systems is $2N - 1$ where $N =\mid S \mid$ denotes the number of states. A simple $8(N - 1)$-competitive online algorithm is the so-called traversal algorithm. The more complicated corresponding work function algorithm for metrical task systems achieves the best possible competitive ratio of $2N - 1$. For details, we refer to Borodin and El-Yaniv [BEY98].

The paging problem and the k-server problem are special cases of metrical task systems. In order to view the k-server problem as a metrical task system, the set of states S is chosen to be the multiset of k points in the metric space. The distance between two states is the minimum distance needed to achieve one server configuration from the other (the minimum weighted matching between the two k-sets). The processing costs for t^i are zero if the request point is in the current k-set. Otherwise, the costs are defined to be infinite.

6.4 Alternative Performance Measures

The concept of competitive analysis can be regarded as a worst-case measure. Like other worst-case measures, for instance the time complexity in the theory of computational complexity, there are some draw-backs with respect to a fair and realistic comparison between two algorithms. As we have seen for the paging problem in the previous sections, the same competitive ratios for LRU and FIFO need not necessarily imply the same performance on practical or random instances. An even worse example in computational complexity is the simplex algorithm for linear programming. In the worst case, the simplex algorithm is exponential. However, for practical problem data it outperforms all known polynomial-time algorithms, like the ellipsoid method or interior point algorithms. Beside reasonable criticism, competitive analysis should be regarded

as it is: a worst-case measure which gives a guaranteed bound on the worst case performance of online algorithms.

As in computational complexity, there are some approaches which deal with some assumptions on the underlying instances. We present two alternative performance measures which are refinements of competitive analysis and are introduced by Koutsoupias and Papadimitriou [KP94].

6.4.1 The Diffuse Adversary

The **diffuse adversary model** [KP94] is based on the assumption of a distribution D which is chosen out of a distribution class $\mathcal{D}$. The correspondent performance ratio of an online algorithm ALG is defined as

$$c_{\mathrm{ALG}}(\mathcal{D}) = \sup_{D \in \mathcal{D}} \frac{E_D[cost_{\mathrm{ALG}}(I)]}{E_D[cost_{\mathrm{OPT}}(I)]}.$$

If we allow $\mathcal{D}$ to be the class of all possible distributions, we do not restrict on special instances and achieve the competitive ratio as introduced in the previous sections.

6.4.2 Comparative Analysis and Lookahead

In comparative analysis [KP94], we choose an online algorithm out of a certain class of online algorithms. We compare this online algorithm with the best algorithm chosen out of another class of algorithms. Let $\mathcal{A}$ be the set of online algorithms and $\mathcal{B}$ the set of possible reference algorithms. Then, we define the **comparative ratio** $c(\mathcal{A}, \mathcal{B})$ as

$$c(\mathcal{A}, \mathcal{B}) = \sup_{B \in \mathcal{B}} \inf_{A \in \mathcal{A}} \sup_{I} \frac{cost_A(I)}{cost_B(I)}$$

The comparative ratio $c(\mathcal{A}, \mathcal{B})$ is the best-possible ratio that an online algorithm of $\mathcal{A}$ may achieve against the best algorithm taken from $\mathcal{B}$. If we assume that $\mathcal{A}$ is the set of all online algorithms and $\mathcal{B}$ is the set of all (offline) algorithms, then $c(\mathcal{A}, \mathcal{B})$ is equal to the usual competitive ratio.

Comparative analysis can be regarded as a two player game. Player $\mathcal{B}$ picks the "best" available algorithm B and player $\mathcal{A}$ chooses an online algorithm A. Then, player $\mathcal{B}$ chooses the instance I in order to maximize the ratio of $\mathcal{A}$'s cost compared to his own cost.

Koutsoupias and Papadimitriou [KP94] examine the comparative ratio for the paging problem and arbitrary metrical task systems. They compare the performance of an online algorithm to the performance of some semi-online algorithms which obtain more information about the input sequence. These algorithms have a **lookahead** of the next l requests. Koutsoupias and Papadimitriou denote the

class of usual online algorithms by $\mathcal{A}_0$ and the class of online algorithms with lookahead l by $\mathcal{A}_l$. They obtain a comparative ratio of $l+1$ for the paging problem. For arbitrary metrical task systems, they achieve a comparative ratio of $2l+1$.

Albers [Alb93a, Alb93b, Alb97b] considers a generalization of online problems and algorithms for the paging problem. She introduces the notions of **lookahead** and **strong lookahead**. Both notions imply that the online algorithm is given more information than in the usual online setting. An online algorithm is said to have lookahead l if it knows the next l requests of the input sequence. It is said to have strong lookahead if it knows the minimal part of the remaining sequence which contains exactly l distinct requests. Albers proves that LRU with strong lookahead l is $(k-l)$-competitive where $l \leq k-2$. She also proves that LRU achieves the best possible competitive ratio under all deterministic online algorithms with strong lookahead l. For more details, we refer to [Alb93a].

Another concept called **max/max-ratio** is introduced by Ben-David and Borodin in [BDB94, BEY98]. In this approach, Ben-David and Borodin compare the worst-case instances of ALG to the worst-case instance of OPT.

6.5 Combinatorial Optimization Problems

Initially, competitive analysis for online problems has been examined for problems arising in computer science. After a couple of years, this technique has been more and more introduced in the analysis of online versions of classical combinatorial optimization problems. In this section, we review some results for the bin packing problem, the scheduling problem for parallel machines, the bipartite matching problem, and two variations of routing problems. We will point out some basic strategies for examining the performance of online algorithms. For a more comprehensive survey on online algorithms for problems arising in computer science and in combinatorial optimization, we refer to [FW98].

6.5.1 Online Bin Packing

The classical (one-dimensional) bin packing problem [GJ79] belongs to the class of $\mathcal{NP}$-hard optimization problems. In this problem, we are given a sequence of items of different size $a_i \in (0,1]$ that have to be packed into unit-size bins of capacity 1. The goal is to find a packing which uses a minimum number of bins. Since the bin packing problem is $\mathcal{NP}$-hard, it is very unlikely to find an optimal solution within polynomial time. Consequently, a lot of research is focused on the development of good approximation algorithms that guarantee a solution within a provable gap from the optimal solution.

For bin packing and a related scheduling problem, which will be considered in Section 6.5.3, Johnson [Joh74] and Graham [Gra69] examined the quality of

online algorithms in comparison to an optimal offline strategy.

In the following, we will introduce some classical approximation algorithms for bin packing. All these algorithms are online algorithms since they assign the items sequentially to the bins.

- **Next Fit (NF)**: Assign the next item to the active bin. If the item does not fit into the active bin, then close this bin and open a new bin that becomes the new active bin.
- **First Fit (FF)**: Assign the next item to the lowest indexed bin into which it will fit. Open a new bin only if the item will not fit into any non-empty bin.
- **Best Fit (BF)**: Assign the next item to the bin into which it fits best, i.e., into the bin that will be maximally filled after the assignment. Open a new bin only if the item will not fit into any non-empty bin.

For NF, FF, and BF, Johnson [Joh74] and Johnson et al. [JDU$^+$74] achieved constant approximation (competitive) ratios. We will present the proof of the result for NF. For the remaining more complicated proofs for FF and BF we refer to Coffman, Garey, and Johnson [CGJ95].

Theorem 6.5.10:

1. NF is 2-competitive
2. FF and BF are $\frac{17}{10}$-competitive

Proof:

1. The lower bound is derived by examining the following input sequence I_{2n} of items with size $a_{2i-1} = \frac{1}{2}$ and $a_{2i} = \frac{1}{2n}$, $i \geq 1$. With every item a_{2i-1}, NF opens a new bin. Consequently, n bins are used for a sequence of $2n$ items. An optimal offline algorithm needs $\lceil \frac{n}{2} \rceil$ bins for the items of size $\frac{1}{2}$ and at most one additional bin for the $\frac{1}{2n}$-size items. We obtain that

$$cost_{\mathrm{NF}}(I_n) \leq \frac{n}{\lceil \frac{n}{2} \rceil} \cdot cost_{\mathrm{OPT}}(I_n) \leq \frac{n}{\frac{n}{2}} \cdot cost_{\mathrm{OPT}}(I_n).$$

   We achieve a competitive ratio of at least 2 for NF.

   The upper bound of 2 can be shown by considering two bins that are closed successively. The total size of the items assigned to this two bins is greater than 1 because NF has opened the second bin. Since the two bins have a capacity of 2, OPT cannot put more than the total size of 2 into these bins. Hence, NF is at most two times worse than OPT.

2. We refer to [JDU+74] and [CGJ95].

❑

A natural polynomial offline heuristic is **First Fit Decreasing (FFD)**. FFD sorts the complete input sequence in decreasing order. Next, FFD assigns the items to the bins in the same way as FF. FFD is shown to be an $\frac{11}{9}$-approximation algorithm.

In contrast to this good offline performance, Van Vliet [Vli92] shows a lower bound of 1.5401 on the competitiveness for online bin packing algorithms. The actual best known algorithm for online bin packing is due to Richey [Ric91]. Richey's **harmonic+1** algorithm achieves a competitive ratio of 1.5888. This algorithm is based on a subdivision of the items into categories according to their weight. The underlying idea is due to Lee and Lee [LL85] who obtained a ratio of 1.69103 for the bounded space bin packing problem which will be considered in the next section. Richey refined the partition into categories by solving several linear programs.

6.5.2 Online Bounded Space Bin Packing

A natural special case of bin packing is the – so called – **bounded space bin packing problem**. In this problem, there are at most a fixed number of bins allowed to be open. The bounded space bin packing problem has many practical applications in which the number of bins that can accessed at the same time is limited. Such a situation occurs for instance in logistic systems where trucks have to be loaded or in communication systems where block-sized units of information have to be transmitted.

We denote by k the number of bins allowed to be open at the same time. We follow a survey of Galambos and Woeginger [GW95].

The k-bounded space version of Next Fit and Best Fit are shown to be nearly $\frac{17}{10}$-competitive. Next-k Fit and Best-k Fit are simple extensions of NF and BF. Next-k Fit and Best-k Fit close always the lowest indexed bin if the current item does not fit in any active bin. The competitive ratios of Next-k Fit and Best-k Fit are slightly greater than $\frac{17}{10}$ but differ from that value only in a small constant depending on k. A modification of Best-k Fit is proposed by Csirik and Johnson [CJ91] who proved that the modified Best-k Fit algorithm BBF-k is $\frac{17}{10}$-competitive, $k \geq 2$. Moreover, they show that for bounded space bin packing their Best Fit strategy BBF-k is better than a First Fit strategy.

Lee and Lee [LL85] proved a lower bound of $h_\infty \approx 1.69103$ for the competitive ratio of the k-bounded space bin packing problem. The value h_∞ is defined as

$$h_\infty := \sum_{i=1}^{\infty} \frac{1}{t_i - 1}$$

where $t_{i+1} = t_i(t_i - 1) + 1$ and $t_1 = 2$. Lee and Lee introduce the **Harmonic** algorithm whose asymptotic competitive ratio is proven to be h_∞. Woeginger [Woe93] achieves the same ratio for his **simplified harmonic** algorithm which simplifies the way of partitioning into the item categories.

6.5.3 Online Scheduling

Graham [Gra66, Gra69] examines the following scheduling problem. A sequence of jobs with different processing times have to be served by m identical parallel machines. The objective is to minimize the last completion time of a job, also called **makespan**. Graham proposed a simple greedy heuristic, called **LIST**, which is an online algorithm. LIST assigns a job always to the least loaded machine, i.e., to the machine that will be finished as the first with serving the jobs assigned to it so far. LIST is shown to be $(2 - \frac{1}{m})$-competitive which is the best possible competitive ratio for $m = 2, 3$ (see Faigle, Kern, and Turan [FKT89]).

For $m = 2$, the general lower bound of $\frac{3}{2}$ for the competitive ratio is easy to derive. We consider a sequence of three jobs. The processing times of the first and of the second job are equal to one. The third job has a processing time of two. An arbitrary online algorithm assigns the first job to an arbitrary machine. If the second job is scheduled on the same machine, a cruel adversary would decide to stop the input sequence. In this case, we would obtain a competitive ratio of 2. If the first two jobs are served on different machines, the third job can be assigned to an arbitrary machine. We achieve a makespan of 3 for the online algorithm whereas an optimal offline algorithm would have produced an schedule with makespan 2 (see Figure 6.5.1). This proves the lower bound of $\frac{3}{2}$.

For $m = 3$, the input sequence proving the lower bound of $\frac{5}{3}$ for the competitive ratio is a little bit more complicated. First, we have three jobs with processing time 1, followed by three jobs with processing time 3, and finally one large job with processing time 6. For this instance, LIST achieves a makespan of 10 compared to an offline optimal makespan of 6 (see Figure 6.5.1).

The upper bound for LIST can be derived in the following way. Let $I = (1, \ldots, n)$ be the input sequence of jobs. We denote by p_i the processing time of job i. The time at which the job is started to be processed by LIST is denoted by s_i. Then, we observe that

$$cost_{\mathrm{OPT}}(I) \geq p_i \quad \forall\, i \qquad \text{and} \qquad cost_{\mathrm{OPT}}(I) \geq \frac{1}{m} \sum p_i$$

The first inequality states that the optimal schedule has a makespan trivially at least as great as each the processing time p_i. The second inequality states that the makespan is at least the average processing time for each machine.

Let us assume that job k is processed as the last. Then, we observe that the starting time s_k of job k is less than or equal to the average processing time so

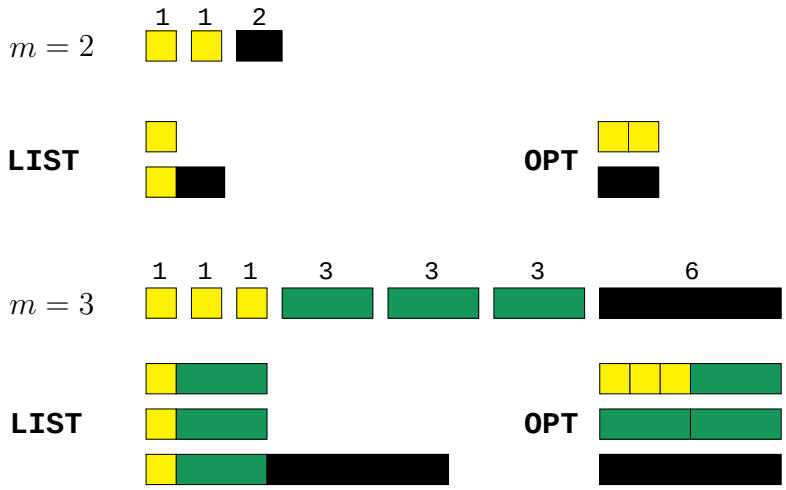

Figure 6.5.1: Instances yielding the lower bound for LIST for 2 and 3 machines.

far, since LIST schedules job k on the least loaded machine. Additionally, no machine is idle at any time between serving two jobs. Hence,

$$s_k \leq \frac{1}{m} \sum_{i \neq k} p_i.$$

Consequently,

$$\begin{aligned} cost_{\text{LIST}}(I) &= s_k + p_k \leq \frac{1}{m} \sum_{i \neq k} p_i + p_k = \frac{1}{m} \sum p_i + \left(1 - \frac{1}{m}\right) p_k \\ &\leq cost_{\text{OPT}}(I) + \left(1 - \frac{1}{m}\right) cost_{\text{OPT}}(I) = \left(2 - \frac{1}{m}\right) cost_{\text{OPT}}(I). \end{aligned}$$

Albers [Alb97a] gives an online scheduling algorithm, called M2, which is 1.923-competitive for m machines where $m \geq 2$. M2 tries to prevent schedules which distribute the load uniformly on all machines. Instead, M2 tries to keep some machines with a "low" load whereas the other machines have a "high" load. Albers [Alb97a] also derives a lower bound of 1.852 on the competitive ratio of deterministic online algorithms. The lower bound holds for $m \geq 80$.

6.5.4 Online Bipartite Matching

The online **bipartite weighted matching problem** in metric spaces is examined for example by Kalyanasundaram and Pruhs [KP91, KP93, KP98]. In the bipartite weighted matching problem, we are given a bipartite graph $G = (S, T, E)$ where S and T denote the bipartition of G, i.e., $E \subseteq S \times T$. We assume that

$| S | = | T |$ and denote by n the cardinality of S and T. For each edge $e \in E$, there is a weight $c_e \in \mathbb{R}_+$. A matching in G is a subset M of n edges of E such that M covers S and T. The objective is to find a matching M for which the total sum of all weights of edges $e \in M$ is minimized. In the following, we assume that we are given a complete bipartite graph ($E = S \times T$). This assumption can be made w.l.o.g. since we can add an edge not in E to G by introducing a weight of at least the sum of all other edge weights. Such an edge will never be chosen in an optimal matching. The bipartite weighted matching problem is often denoted as the **assignment problem**.

Kalyanasundaram and Pruhs [KP93] consider the online bipartite matching problem for some given metric space $\mathcal{M}$. Each node of the graph corresponds to a point in $\mathcal{M}$. The weight c_e of an edge $e \in E$ corresponds to the distance of the end nodes of e in $\mathcal{M}$. Consequently, the edge weights c_e are non-negative and satisfy the triangle inequality. The metric space $\mathcal{M}$ and the set S are known to the online algorithm in advance. The set T is revealed to the online algorithm vertex by vertex. With each vertex $v \in T$, the weights (distances) c_e for all (u, v) with $u \in S$ are given to the online algorithm. The online algorithm has to choose an edge (u, v) among all these edges before the next vertex of T is revealed. The edge chosen by the online algorithm becomes a matching edge.

Kalyanasundaram and Pruhs [KP93] show that for every n there is a finite metric space $\mathcal{M}$ for which the competitive ratio of deterministic online algorithms is bounded from below by $2n - 1$ (cf. Figure 6.5.2). This finite metric space corresponds to a complete graph $G' = (V = \{z, u_1, u_2, \ldots, u_n\}, E')$ with a vertex z whose distance to every other vertex u_i is equal to one. The distance between u_i and u_j, $i \neq j$, is two. Edges (u_i, u_i) have a weight of zero. The associated bipartite graph is $G'' = (V, V, E')$. Let ALG be an arbitrary deterministic online algorithm for the bipartite weighted matching problem.

The instance which proves the lower bound of $2n-1$ for the competitive ratio is constructed in the following way (cf. Figure 6.5.2). S is chosen to be $V \backslash \{z\}$. T is a n-size subset of V containing z and is constructed depending on the choices of ALG. We consider the bipartite graph $G = (S, T, E)$ which is a subgraph of G''. Initially, T contains only z. This means that z is given to ALG as first. All distances from a vertex of S to z are equal to one. By u_i, we denote the vertex of S that is adjacent to the edge that ALG adds to the matching, i.e., ALG chooses (u_i, z). Then, u_i is added to T and presented to ALG in the next step. Any vertex of S, except of u_i, has distance 2 from u_i. Since ALG has to choose an edge connecting one of these vertices to u_i, the weight of the matching produced by ALG increases by two in this step. Let (u_i, u_j) be the edge chosen by ALG. Once again, the corresponding vertex u_j, $j \neq i$, is added to T. This procedure is repeated until T contains n vertices. The resulting matching has a total weight of $2n - 1$, since ALG always has to choose an edge of weight two in step i, $2 \leq i \leq n$.

In contrast to ALG, an optimal offline algorithm OPT knows S and T a-

priori. OPT assigns the unique vertex $u_k \in S \backslash T$ to z. The edge (u_k, z) has a weight of one. Additionally, OPT chooses the edges (u_i, u_i) with weight zero for all other nodes of S. Consequently, the optimal matching has a total weight of one. We obtain that the competitive ratio of any arbitrary deterministic online algorithm is at least $2n - 1$.

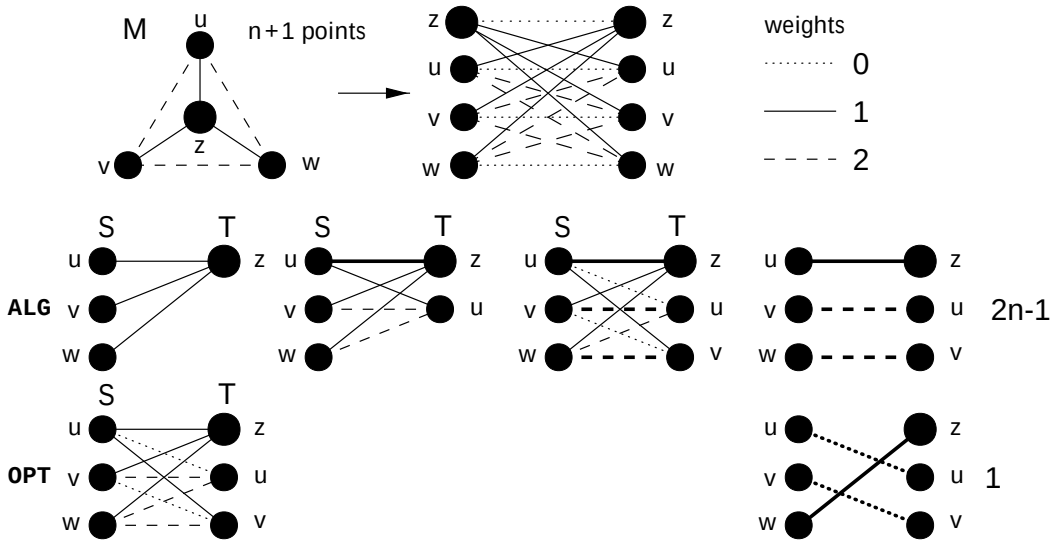

Figure 6.5.2: Instance of n nodes for which any deterministic online algorithm yields a matching with a weight of at least $2n-1$ whereas an optimal offline matching has weight 1.

Kalyanasundaram and Pruhs [KP93] prove that the greedy strategy (NEAREST NEIGHBOR) choosing in each step the edge of minimum weight is only $(2^n - 1)$-competitive. They present a $(2n-1)$-competitive deterministic online algorithm, called PERMUTATION. In each step i, PERMUTATION either chooses an edge of weight zero or computes a partial matching of minimum weight. This partial matching M_i matches all vertices of T known so far and the vertex of T revealed in the actual step is matched to that unique vertex $s_i \in S$ covered by $M_i \backslash M_{i-1}$.

Moreover, Kalyanasundaram and Pruhs [KP93] consider the online problem of computing a maximal weighted bipartite matching. They prove that the corresponding greedy algorithm, called FARTHEST NEIGHBOR, achieves a matching with a weight of at least $\frac{1}{3}$ the maximum weight offline perfect matching.

Khuller, Mitchell, and Vazirani [KMV94] independently obtain the same results as Kalyanasundaram and Pruhs for the online minimum weight bipartite matching problem. Additionally, they consider the online stable marriage problem where the number of unstable pairs should be minimized. In the stable

marriage problem, for each vertex of T we are given a preference list (ranking) of vertices in S and for each vertex of S we are given a preference list of vertices in T. A pair of vertices (u, v), $u \in T$ and $s \in S$, is said to be unstable if there are two other vertices $u' \in T$ and $v' \in S$ where u prefers v' and v prefers u'. Khuller et al. show that the deterministic "first come, first serve"-strategy yields $O(n \log n)$ unstable pairs on the average. They also give a lower bound of $\Omega(n^2)$ on the worst-case performance of any deterministic and any randomized online algorithm. Note that a stable marriage can always be found offline [KMV94], for instance in $O(n^2)$ by applying the Gale-Shapley algorithm [Knu96, GI89].

Karp, Vazirani, and Vazirani [KVV90] present an optimal algorithm for the online bipartite cardinality matching problem. Given a not necessarily complete bipartite graph where the vertices of the second bipartition and the corresponding edges are revealed online, the objective is to compute a matching of maximal cardinality. Karp et al. prove that the expected size of the matching produced by the introduced online algorithm is at least $(1 - \frac{1}{e})m + o(m)$ where m is the size of the maximal offline matching. Moreover, they show that this is (up to some lower order terms) the best possible performance of an online algorithm. The performance measure used by Karp et al. differs from the usual competitive ratio. Karp et al. define the performance of an online algorithm ALG as the minimum expected size of the matching achieved by ALG where the expectation is taken according to the randomization of ALG. This minimum is computed over all possible bipartite graphs G and all possible input sequences of the second bipartition.

6.5.5 Online Routing

Routing problems have a large spectrum of applications in robotics and transportation. In this section, we consider the online routing problem of a server in an underlying metric space. A request consists of a point in the metric space to which the server has to move after the request is released. We distinguish between the routing problem in which the server is not forced to return to its initial position after serving all the requests and the traveling salesman problem (TSP) in which the server has to move back to the origin. Both online versions have been examined by Ausiello et al. in [AFL$^+$94, AFL$^+$95].

First, we consider the routing problem. By $\mathcal{M}$, we denote the underlying metric space. The i-th request consists of a pair (p_i, r_i) where p_i is a point in $\mathcal{M}$ and r_i denotes the release time. This online situation differs from the common online setting where the next request is revealed just after the previous one is processed. Here, each request is revealed at a time r_i while the online algorithm moves the server. The corresponding offline problem can be solved in quadratic time for the real line [PSMK90]. For other metric spaces, it turns out to be $\mathcal{NP}$-hard [KNI93]. Ausiello et al. [AFL$^+$94] show that neither a deterministic nor a randomized online algorithm can achieve a better competitive ratio than

2. They introduce a "greedy" strategy computing at every moment the shortest Hamiltonian path connecting all request points known so far. This Hamiltonian path starts at the current server position and visits all points corresponding to the requests that are not yet served. The greedy strategy is proven to be $\frac{5}{2}$-competitive. The major drawback of this strategy is that its time complexity is exponential. Therefore, Ausiello et al. [AFL+94] consider a heuristic based on the minimum spanning tree heuristic for the traveling salesman problem which is known to be a 2-approximation algorithms for the TSP with distances satisfying the triangle inequality. The corresponding online algorithm MST is shown to be 3-competitive. For the special case of the real line, Ausiello et al. [AFL+94] develop an online algorithm, called SNB (START NEAR THE BEGINNING), which moves always to one of the extreme points of the interval containing all the requests points. SNB chooses the extreme point that is located the nearest to the origin. Ausiello et al. [AFL+94] prove that SNB is $\frac{7}{3}$-competitive for the real line.

For the online traveling salesman problem, Ausiello et al. [AFL+95] derive a lower bound of $(9 + \sqrt{17})/8$ on the competitiveness for any arbitrary online algorithm. Once again, the greedy algorithm computing the shortest tour visiting all request points is $\frac{5}{2}$-competitive. Ausiello et al. [AFL+95] introduce a 2-competitive online algorithm called Plan-at-home (PAH). PAH works as follows:

1. If the server is at the origin, it starts to follow an optimal route that visits all request points and goes back to the origin.

2. If a new request is presented to PAH, then it proceeds with one of the following two actions depending on the current server position p:

   (a) If the distance from the new request point to the origin is greater than the distance from the current position back to the origin, the server is moved back to the origin. Then, PAH proceeds with step 1.

   (b) Otherwise, the new request is ignored. The server proceeds to follow the current route. The new request is included when the server arrives back at the origin.

For the special case of the real line, Ausiello et al. [AFL+95] present an online algorithm that is $\frac{7}{4}$-competitive.

Chapter 7

Online Tram Dispatch

In this chapter, we examine the competitiveness of online versions of the following tram dispatching problems: the tram dispatch problem (TDP), the type mismatch problem (TMP), the departure type dispatch problem (DTDP), and the type mismatch problem at departure (DTMP).

For these problems, the performance of online algorithms is analyzed in different online settings. The online settings differ in the information which is available for the online algorithms.

Before we examine the different online situations for the tram dispatching problems, we introduce a modified notion of competitiveness, called $(\mathbf{c}, \mathbf{d})$-**competitiveness**. This notion is motivated by the observation that we often achieve a shunting-free and type-preserving solution for the different tram dispatching problems.

In the following, we derive lower and upper bounds on the competitiveness of the different tram dispatching problems. We start with the combined arrival and departure problem (see Section 7.2). For the shunting problem (TDP), we derive a lower and an upper bound on the (extended) competitive ratio of any deterministic online algorithm. Then, we consider the online type mismatch problem (TMP). For the online departure problems DTDP and DTMP, we derive lower bounds on the performance of any arbitrary deterministic online algorithm. We discuss the performance of a class of online algorithms for DTMP. We conclude each subsection with some remarks on the performance of randomized online algorithms for the considered online tram dispatching problem.

7.1 $(\mathbf{c}, \mathbf{d})$-Competitiveness

For the tram dispatching problems, we observed that optimal solutions often do not require shunting movements or type mismatches. The definition of competitiveness (cf. Definition 6.1.1) implies that for such instances with an optimal cost value of zero, the cost of any competitive online algorithm must not be greater

than a constant independent of the size of the instance.

As we have seen in the example of the online weighted matching problem (cf. Section 6.5.4), some online problems (and algorithms for these problems) have a competitive ratio that is linear in the size of the input. In this chapter, we will observe that the same holds for the tram dispatch problems. We will introduce several problem instances for which any arbitrary deterministic online algorithm needs shunting movements or type mismatches at least in the order of the number of trams whereas an optimal solution is shunting-free and type preserving. Consequently, none of these problems and no deterministic online algorithm for these problems is competitive in the sense of Definition 6.1.1, because b is not constant.

For this reason, we extend the notion of competitiveness in the following way. We split the examination of the performance of an online algorithm ALG into two parts. First, we consider the performance on instances for which an optimal offline algorithm OPT achieves a solution with non-zero cost. Secondly, we compute the worst-case cost obtained by ALG for instances for which OPT has zero cost. We call this performance measure $(\mathbf{c}, \mathbf{d})$**-competitiveness**. Analogously to Definition 6.1.1, we introduce the following definition:

Definition 7.1.1: A deterministic online algorithm ALG is said to be $(\mathbf{c}, \mathbf{d})$**-competitive**, if and only if the following inequalities hold for all problem instances I

$$cost_{\mathrm{ALG}}(I) \leq \begin{cases} c \cdot cost_{\mathrm{OPT}}(I) & \text{if } cost_{\mathrm{OPT}}(I) \neq 0, \\ d & \text{if } cost_{\mathrm{OPT}}(I) = 0 \end{cases}$$

where $cost_{\mathrm{ALG}}(I)$ denotes the cost of the solution produced by ALG for the instance I, $cost_{\mathrm{OPT}}(I)$ is the optimal cost computed by an optimal algorithm OPT knowing the complete instance I in advance. Analogously to Definition 6.1.1, OPT is called **optimal offline algorithm**. ❑

For randomized online algorithms and oblivious adversaries, we introduce the following analogous definition:

Definition 7.1.2: A randomized online algorithm RAND is said to be $(\mathbf{c}, \mathbf{d})$**-competitive** against any oblivious adversary if and only if the following inequalities hold for all problem instances I

$$E[cost_{\mathrm{RAND}}(I)] \leq \begin{cases} c \cdot cost_{\mathrm{OPT}}(I) & \text{if } cost_{\mathrm{OPT}}(I) \neq 0, \\ d & \text{if } cost_{\mathrm{OPT}}(I) = 0 \end{cases} .$$

Here, $E[cost_{\mathrm{RAND}}(I)]$ denotes the expected cost of RAND where the expectation is taken over the probability distribution corresponding to all internal coin flips of RAND. $cost_{\mathrm{OPT}}$ denotes the optimal cost computed by an optimal algorithm

OPT knowing the instance in advance. The best possible that an oblivious adversary can do is to use OPT for computing a solution for the chosen instance I. ❑

7.2 The Arrival-Departure-Problem

We split the examination of the online TDP into two parts. We start with considering instances of TDP for which an optimal offline algorithm yields a shunting-free and type-preserving solution. Then, we examine the performance of deterministic online algorithms for instances for which an optimal offline algorithms needs at least one shunting movement.

By $\mathcal{I}_0$, we denote the set of instances of TDP for which an optimal offline algorithms yields an assignment of zero cost. The set of the remaining instances is denoted by $\mathcal{I}_1$.

Before we start with these investigations, we define the online setting for TDP.

> Given R stacks of length P_r, $1 \leq r \leq R$, with N positions in $\mathcal{P}$ and a departure sequence $\mathcal{D}$ of N trams with types in $\mathcal{T}$, the arrival sequence $\mathcal{A}$ of N trams is revealed to the online algorithm ALG tram by tram. After ALG has assigned the actual tram, the type of the next tram is presented to ALG. The tram types of $\mathcal{A}$ are chosen in such a way that
>
> $$| \{a \in \mathcal{A} \mid t(a) = \tau\} | = | \{d \in \mathcal{D} \mid t(d) = \tau\} | \quad \text{for each} \quad \tau \in \mathcal{T}.$$

This setting corresponds to the practical real-time version of TDP. Usually, the actual arrival sequence differs substantially from the planned arrival sequence. Hence, one way to examine the real-time events is to assume that the only information available for the dispatcher is the information of the next arrival.

7.2.1 Performance on $\mathcal{I}_0$

First, let us consider an instance of the TDP in which a fixed departure sequence $\mathcal{D}$ is given. For each departure $d \in D$, a type $t(d)$ is specified. Additionally, we know the sizes of the stacks $\mathcal{P}_r$ for all $1 \leq r \leq R$.

Theorem 7.2.3: For any online algorithm ALG, there is an instance of $\mathcal{I}_0$ for which ALG requires shunting whereas an optimal offline algorithm OPT yields a solution without shunting.

Proof: We prove the theorem by constructing an arrival sequence $\mathcal{A}$ for which any online algorithm needs at least one shunting movement.

We consider the following instance of the TDP. Let $\mathcal{D} = \{d_1, d_2, d_3, d_4, d_5\}$ be the set of departures. The required types are given by the mapping $t : \mathcal{D} \to \mathcal{T}$,

where $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, $t(d_1) = t(d_4) = \tau_1$, $t(d_2) = t(d_5) = \tau_2$, and $t(d_4) = \tau_3$. We define the set of arrivals $\mathcal{A} = \{a_1, \ldots, a_5\}$. The existence of an assignment fulfilling the type requirements implies that there must be two arrivals having type τ_1, two arrivals having type τ_2 and exactly one arrival of type τ_3.

The depot to which the trams are to be assigned consists of two stacks (cf. Figure 7.2.1). The first stack $\mathcal{P}_1$ has a length of 3 and the second stack $\mathcal{P}_2$ a length of 2. The positions in stack $\mathcal{P}_1$ are numbered consecutively from p_1 to p_3 beginning at the bottom. Analogously, the positions in stack $\mathcal{P}_2$ are denoted by p_4 and p_5.

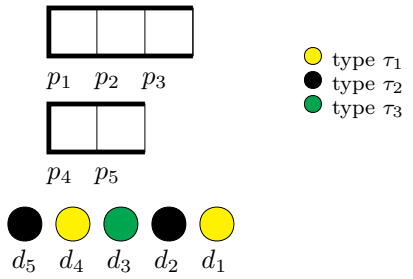

Figure 7.2.1: The information given in advance to the online algorithm.

This information is known to every online algorithm and, of course, to every offline algorithm.

We start constructing the set $\mathcal{A}$ by defining $t(a_1) = \tau_1$. The tram a_1 has to be assigned immediately to a stack position.

By Theorem 4.1.7, shunting is only required either for assigning the arriving trams or for their departure. Without loss of generality, we assume that shunting occurs only between trams leaving the depot.

Consequently, each online algorithm has to assign tram a_1 to a position without forcing a shunting movement for any tram a_i, $i > 1$.

Since there are as many arriving vehicles as stack positions, the only positions to which a_1 can be assigned are the bottom positions p_1 and p_4.

Case 1: We assume that the online algorithm assigns a_1 to p_1 (cf. Figure 7.2.2). Then, tram a_2 of type τ_1 is given to the online algorithm. This tram can only be assigned to position p_2 or p_4. Both alternatives lead to a free stack position on top of a tram of type τ_1. Since $| \mathcal{A} | = | \mathcal{P} |$, a tram of a different type must be assigned to this free position. Since $t(d_1) = \tau_1$, at least one shunting movement is necessary for the corresponding departure of a tram of type τ_1. If we define $t(a_3) = t(a_5) = \tau_2$ and $t(a_4) = \tau_3$, an optimal offline algorithm yields a solution where no shunting movement is required (cf. Figure 7.2.2). Hence, if an online algorithm assigns a_1 to position p_1, at least one shunting movement is required in contrast to an optimal offline solution where shunting is unnecessary.

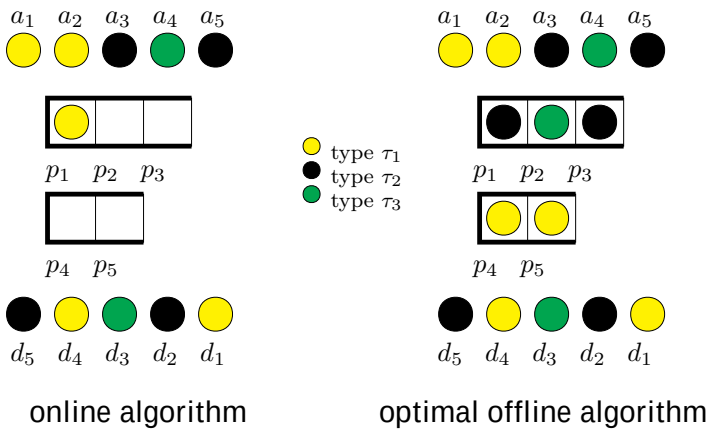

Figure 7.2.2: Comparison of the assignments achieved by the arbitrary online algorithm and an optimal offline algorithm (Case 1).

Case 2: We assume that a_1 is assigned to position p_4. As the next arriving tram a_2, a tram of type τ_3 is chosen. Once again, any online algorithm has only two choices to assign a_2: position p_1 or position p_5.

Case 2.1: If the online algorithm assigns a_2 to position p_5 (cf. Figure 7.2.3), then the next tram a_3 is chosen to be of type τ_1. Thus, the online algorithm must assign a tram of type τ_1 to a bottom position. We complete the arrival sequence by choosing a_4 and a_5, both of type τ_2. As in the first case, the online algorithm needs a shunting movement for the departure. If we apply an optimal offline algorithm to this arrival sequence, we achieve a solution without shunting (cf. Figure 7.2.3).

Case 2.2: We suppose that tram a_1 of type τ_1 is assigned to p_4 and tram a_2 of type τ_3 is assigned to p_1. The third and fourth arriving trams a_3 and a_4 are of type τ_2. The type of the last tram a_5 is τ_1 (cf. Figure 7.2.4).

Then, a tram of type τ_2 has to be assigned to position p_2. The remaining two possibilities for the online algorithm to assign the trams of type τ_1 and τ_2 are illustrated in Figure 7.2.4. In both cases, one shunting movement is required for the departure. For the same arrival sequence, an optimal offline algorithm yields a solution without shunting.

Consequently, for this particular departure sequence $\mathcal{D}$ and depot positions $\mathcal{P}$ as well as for any arbitrary online algorithm ALG, we can construct an arrival sequence $\mathcal{A}(\text{ALG})$ for which an assignment determined by ALG needs at least one shunting movement. In contrast, an optimal offline algorithm applied to $\mathcal{A}(\text{ALG})$ always yields an assignment that avoids shunting completely. ❑

Theorem 7.2.3 implies that for every online algorithm ALG we can construct

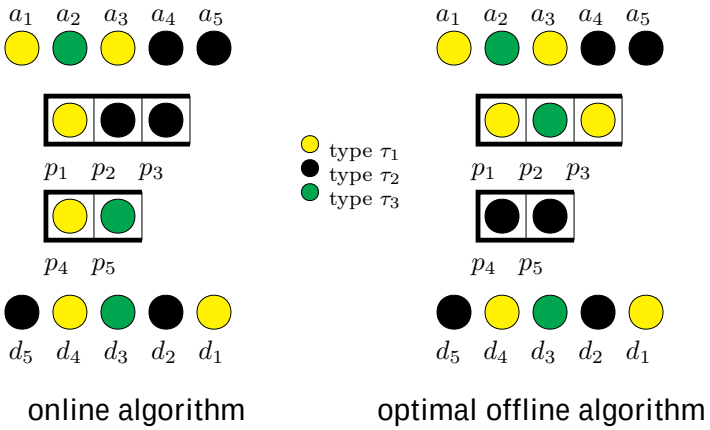

Figure 7.2.3: Comparison of the best online and an optimal offline assignment for Case 2.1.

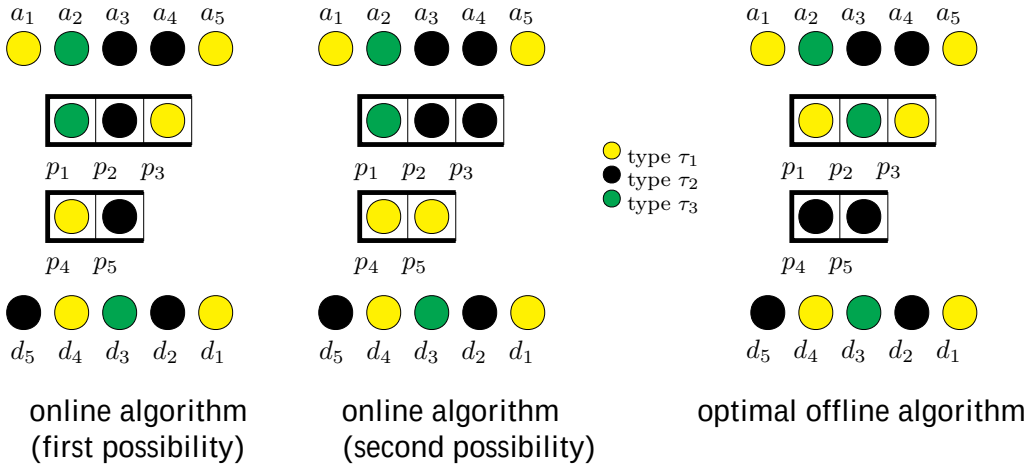

Figure 7.2.4: Comparison of the online and offline assignments for Case 2.2.

an arrival sequence so that ALG behaves worse than an optimal offline algorithm. The next theorem gives a lower bound for the competitiveness of every online algorithm for instances in $\mathcal{I}_0$.

Theorem 7.2.4: Every online algorithm yields a solution of at least $\frac{N}{3}$ shunting movements for a class of instances of $\mathcal{I}_0$ where $N = 6l$ denotes the number of trams, $l \geq 2$.

Proof: We prove the theorem by constructing instances $(\mathcal{A}_i, \mathcal{D}, \mathcal{P})$, $i = 1, 2$ for which OPT determines an assignment that avoids shunting. We show that any (arbitrary) online algorithm ALG yields an assignment which needs at least

$\min\{l^2, 2l\}$ shunting movements where $l = \frac{N}{6}$.

We construct two arrival sequences $\mathcal{A}_1$ and $\mathcal{A}_2$, each of length $N = 6l$, $l \geq 1$, and a departure sequence $\mathcal{D}$ of length $M = N$. The depot in which the arriving trams will be stored consists of two stacks $\mathcal{P}_1$ and $\mathcal{P}_2$. Stack 1 is of length $4l$ and stack 2 of length $2l$. The positions of stack 1 are numbered consecutively from 1 to $4l$ where p_1 denotes the bottom position. Analogously, we number the positions of stack 2 from $4l+1$ to $6l$ where p_{4l+1} denotes the bottom position.

The types of the departures $d_1, d_2, \ldots, d_N$ are defined as

$$t(d_i) = \begin{cases} \tau_1 & \text{if} \quad 1 \leq i \leq l \quad \vee \quad 4l+1 \leq i \leq 5l \\ \tau_2 & \text{if} \quad l+1 \leq i \leq 2l \quad \vee \quad 5l+1 \leq i \leq 6l \\ \tau_3 & \text{if} \quad 2l+1 \leq i \leq 4l \end{cases} .$$

Next, we construct two arrival sequences $\mathcal{A}_1 = \{a_1^1, a_2^1, \ldots, a_N^1\}$ and $\mathcal{A}_2 = \{a_1^2, a_2^2, \ldots, a_N^2\}$. The corresponding types of the arriving trams are given by

$$t(a_i^1) = \begin{cases} \tau_1 & \text{if} \quad 1 \leq i \leq 2l \\ \tau_2 & \text{if} \quad 2l+1 \leq i \leq 3l \quad \vee \quad 5l+1 \leq i \leq 6l \\ \tau_3 & \text{if} \quad 3l+1 \leq i \leq 5l \end{cases} .$$

and

$$t(a_i^2) = \begin{cases} \tau_1 & \text{if} \quad 1 \leq i \leq l \quad \vee \quad 3l+1 \leq i \leq 4l \\ \tau_2 & \text{if} \quad 4l+1 \leq i \leq 6l \\ \tau_3 & \text{if} \quad l+1 \leq i \leq 3l \end{cases} .$$

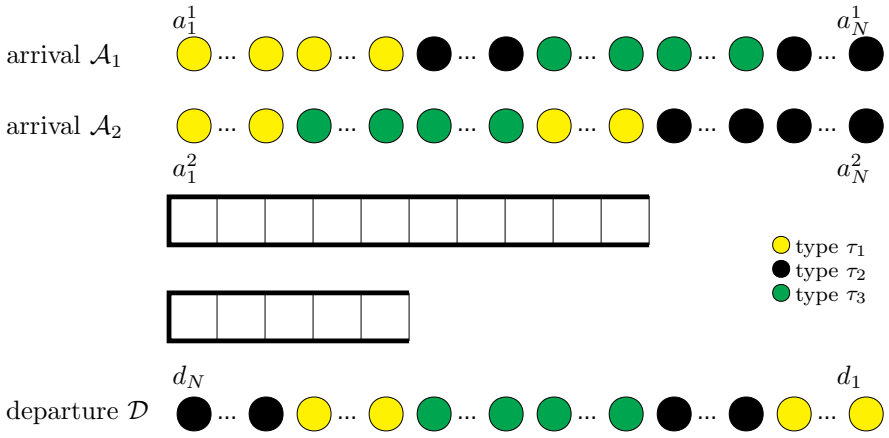

Figure 7.2.5: The arrival and departure sequences.

In both sequences $\mathcal{A}_1$ and $\mathcal{A}_2$, the first arriving tram is of type τ_1. ALG has two alternatives to assign this tram: ALG can assign a_1^1 (resp. a_1^2) either to the bottom position p_1 of stack 1 or to the bottom position p_{4l+1} of stack 2.

In the first case, we choose $\mathcal{A}_1$ as the sequence to be served by ALG. In the second case, we choose $\mathcal{A}_2$. We examine both cases separately.

Case 1: We assume that a_1^1 is assigned to position p_1. Next, ALG has to assign $2l-1$ trams of type τ_1, followed by l trams of type τ_2, $2l$ trams of type τ_3, and the remaining l trams of type τ_2.

For each arriving tram, ALG has two possibilities to assign it: either it is assigned to stack 1 or to stack 2. In both cases, the position to which the tram is assigned is the position p_i with minimum index i amongst all empty positions in the corresponding stack. Consequently, we get an assignment of trams (tram types) to positions as follows:

position index (begin — end)			stack 1	number of assigned trams
			top	
$5l-j-k-o+1$	to	$4l$	type τ_2	$-l+o+j+k \geq 0$
$3l-j-k+1$	to	$5l-j-k-o$	type τ_3	$2l-o \geq 0$
$2l-k+1$	to	$3l-j-k$	type τ_2	$l-j \geq 0$
1	to	$2l-k$	type τ_1	$2l-k \geq 1$
			bottom	

position index (begin — end)			stack 2	number of assigned trams
			top	
$k+j+o+1$	to	$2l$	type τ_2	$2l-o-j-k \geq 0$
$k+j+1$	to	$k+j+o$	type τ_3	$o \geq 0$
$k+1$	to	$k+j$	type τ_2	$j \geq 0$
1	to	k	type τ_1	$k \geq 0$
			bottom	

In stack 1, the trams of type τ_1 are at the bottom positions. We denote by k, $0 \leq k \leq 2l-1$, the number of trams of type τ_1 assigned to stack 2. If $k > 0$, these trams are assigned to positions p_{4l+1} to p_{4l+k} at the bottom of stack 2. By j and o, we denote the number of trams of type τ_2 and τ_3 are assigned to stack 2 where the o trams of type τ_3 are stored on top of the j trams of type τ_2. Note that $0 \leq j \leq l$ and $0 \leq o \leq 2l$. The sum of o, j, and k is greater than or equal to l and must not exceed $2l$.

We introduce the following four non-negative variables u, v, c, and d describing the possibilities of the online algorithm to assign the trams of type τ_2 stored in stack 1 and 2 (cf. Figure 7.2.6). The variable u denotes the number of trams of type τ_2 which are stored in stack 1 between trams of type τ_1 and (the possibly assigned) trams of type τ_3 and which are assigned to the last block of departures of type τ_2. By c, we denote the number of trams of type τ_2 which are assigned to

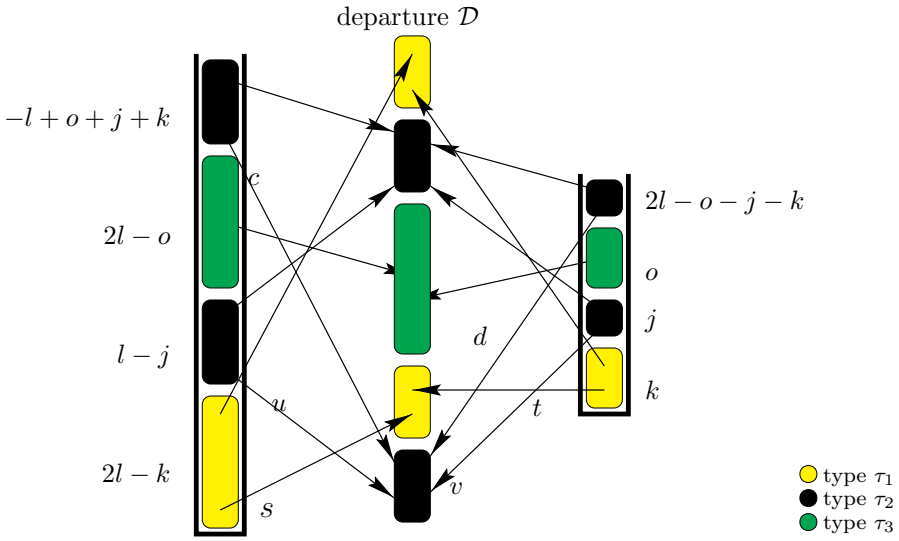

Figure 7.2.6: Possible assignment of arrivals to positions and to departures (for arrival sequence $\mathcal{A}_1$).

the same set of departures and which are stored at the top positions of the same stack. Analogously to u and c, we define v and d for stack 2. Note that the sum of u, v, c, and d must be equal to l so that the last l departures of type τ_2 are covered.

Furthermore, we define s to be the number of trams of type τ_1 in stack 1 which are assigned to the last l departures of type τ_1. Analogously, t is defined as the number of trams of type τ_1 in stack 2 assigned to the remaining $l-s$ departures of the same block of type τ_1. Note that $s+t=l$.

Using these notations, the number of shunting movements required for the assignment determined by the online algorithm can be expressed by the following function g. The first part of the sum corresponds to the shunting movements required for stack 1 and the second part to the shunting movements in stack 2. Note that all terms of g are non-negative.

$$
\begin{aligned}
g(o,j,k,s,t,u,v,c,d) \;:=\;& u\,s + c\,s + u\,(2l-k-s) + (l-j-u)\,(2l-k-s) \\
& +(2l-o)\,(2l-k-s) + c\,(2l-k-s) \\
& +(-l+o+j+k-c)\,(2l-k-s) \\
& +(2l-o)\,(l-j-u) + c\,(l-j-u) + c\,(2l-o) \\
+\;& v\,t + d\,t + v\,(k-t) + (j-v)\,(k-t) + o\,(k-t) \\
& +d\,(k-t) + (2l-o-j-k-d)\,(k-t) + o\,(j-v) \\
& +d\,(j-v) + d\,o
\end{aligned}
$$

where

$$\left.\begin{array}{llll} 0 \le c \le & -l+o+j+k & 0 \le d \le & 2l-o-j-k \\ 0 \le u \le & l-j & 0 \le o \le & 2l \\ 0 \le s \le & 2l-k & 0 \le j \le & l \\ & & 0 \le v \le & j \\ & & 0 \le k \le & 2l-1 \\ & & 0 \le t \le & k \\ s+t = l = & u+v+c+d & & \\ (o,j,k,s,t,u,v,c,d \text{ integer}) & & & \end{array}\right\} \quad (7.2.1)$$

We define $G(l)$ a set the of all feasible solutions $x = (o, j, k, s, t, u, v, c, d)$ according to (7.2.1) for any fixed l.

In the following, we give a feasible solution $x^* \in G(l)$ whose value $g(x^*)$ is equal to $2l$. Then, we prove that g is bounded from below by $2l$ on $G(l)$ for all $l \ge 2$. If $l = 1$, g is bounded from below by l^2.

We consider the following feasible vector $x^* = (o^*, j^*, k^*, s^*, t^*, u^*, v^*, c^*, d^*) = (0, 0, 2l-1, 1, l-1, l, 0, 0, 0)$. The number of shunting movements is given by $g(x^*) = 2l$. The first term us and the term $(2l-o-j-k-d)(k-t)$ of g are equal to l whereas the other terms are equal to zero.

Next, we substitute t by $l - s$. We sum up the terms in $2l-k-s$ and in $k-l+s$. All terms of g stay non-negative.

$$\begin{array}{rcl} g(o,j,k,s,t,u,v,c,d) & = & u\,s + c\,s + (2l-k-s)\,(2l+k) + (2l-o)\,(l-j-u) \\ & & +c\,(l-j-u) + c\,(2l-o) \\ & + & v\,(l-s) + d\,(l-s) + (k-l+s)\,(2l-k) \\ & & +o\,(j-v) + d\,(j-v) + d\,o \end{array}$$

Case 1.1: If $2l-k-s \ge 1$, then g is bounded from below by $2l$, because $k \ge 0$.

Case 1.2: If $2l-k-s = 0$, $k+s$ is equal to $2l$. This implies that $s \ge 1$ and $k \ge l$ because $k \le 2l-1$ and $s \le l$.

This leads to

$$\begin{array}{rcll} g(o,j,k,s,t,u,v,c,d) & \ge & (u+c)\,s + (v+d)\,(l-s) + (k-l+s)\,(2l-k) & \\ & = & (u+c)\,s + (v+d)\,(l-s) + l\,(2l-k) & (7.2.2) \end{array}$$

Case 1.2.1: If $s < l$, then $s \ge 1$ and $(l-s) \ge 1$. Since $u+c+v+d = l$, the sum of the first terms in (7.2.2) is at least l. The last term is at least l because $k \le 2l-1$. Therefore, g is bounded from below by $2l$.

Case 1.2.2: If $s = l$, then $k = l$ and g is bounded from below by l^2 which is at least $2l$ for $l \ge 2$.

Summarizing the discussion, ALG needs at least $\min\{l^2, 2l\}$ shunting movements in order to serve $\mathcal{A}_1$ whereas an optimal offline algorithm OPT avoids

shunting completely. An assignment determined by an optimal offline algorithm OPT can be described as follows. OPT assigns all trams of type τ_1 to stack 2. The remaining trams are assigned to stack 1. All the trams enter the depot and leave it in the opposite order of their arrival without shunting.

Case 2: We consider the second case in which the first tram is assigned to stack 2. In this case, the online algorithm ALG has to serve the arrival sequence $\mathcal{A}_2$.

The first l arriving trams in $\mathcal{A}_2$ are of type τ_1. Then, $2l$ trams of type τ_3 arrive, followed by l trams of type τ_1. Finally, $2l$ trams of type τ_2 are to be assigned. Analogously to the first case, we obtain the following assignment of trams (tram types) to the stacks:

position index (begin — end)			stack 1	number of assigned trams
			top	
$4l-j-k-o+1$	to	$4l$	type τ_2	$o+j+k \geq 0$
$3l-k-o+1$	to	$4l-j-k-o$	type τ_1	$l-j \geq 0$
$l-k+1$	to	$3l-k-o$	type τ_3	$2l-o \geq 0$
1	to	$l-k$	type τ_1	$l-k \geq 0$
			bottom	

position index (begin — end)			stack 2	number of assigned trams
			top	
$k+j+o+1$	to	$2l$	type τ_2	$2l-o-j-k \geq 0$
$k+o+1$	to	$k+j+o$	type τ_1	$j \geq 0$
$k+1$	to	$k+o$	type τ_3	$o \geq 0$
1	to	k	type τ_1	$k \geq 1$
			bottom	

An assignment of these trams to the departures of $\mathcal{D}$ is illustrated in Figure 7.2.7.

Once again, shunting is not necessary for the assignment of the homogeneous blocks of each type, i.e., blocks of consecutive positions to which only trams of the same type are assigned.

The number of shunting movements depends on the values of o, j, and k where o denote the number of trams of type τ_3 that are assigned to stack 2 and k and j denote the number of trams of the first and the second block of arriving trams of type τ_1. By c and d, we denote the number of trams of type τ_2 that are assigned to the last block of type τ_2 trams in the departure sequence and are stored in stack 1 or 2, respectively. By s and t, we denote how many trams of type τ_1 arriving as first are assigned to stack 1 and 2, respectively. Analogously, u and v denote the number of the last l arriving trams of type τ_1 assigned to

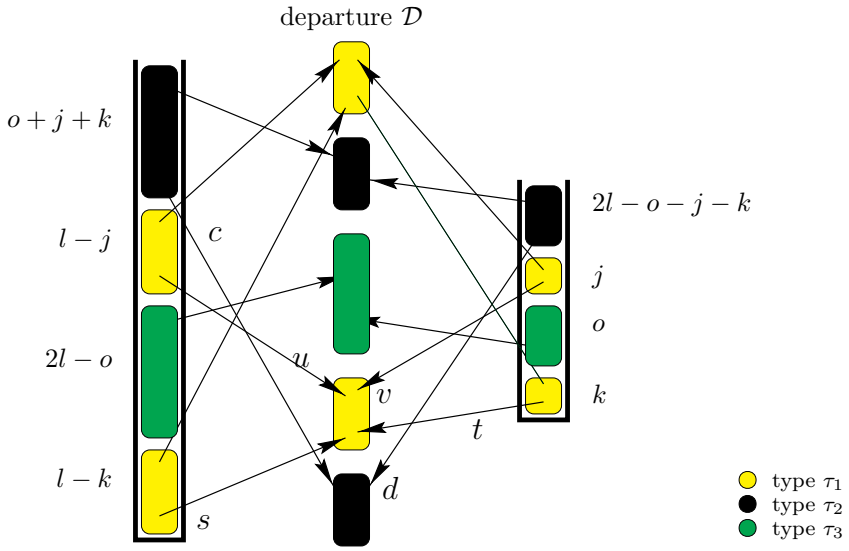

Figure 7.2.7: Possible assignment of arrivals to positions and to departures (for arrival sequence $\mathcal{A}_2$).

stack 1 or 2. These $s+t+u+v$ trams serve the departures $d_{4l+1}, \ldots, d_{5l}$.

Overall, the number of shunting movements necessary for the departure is given by the following function in o, j, k, s, t, u, v, c, and d:

$$\begin{aligned} g(o,j,k,s,t,u,v,c,d) \;:=\; & s\,c + (l-k-s)\,(2l-o) + (l-k-s)\,u + (l-k-s)\,c \\ & +(l-k-s)\,(o+j+k-c) + (2l-o)\,u + (2l-o)\,c \\ & +u\,c + (l-j-u)\,c + (l-j-u)\,(o+j+k-c) \\ +\; & t\,d + (k-t)\,o + (k-t)\,v + (k-t)\,d \\ & +(k-t)\,(2l-o-j-k-d) + o\,v + o\,d + v\,d \\ & +(j-v)\,d + (j-v)\,(2l-o-j-k-d) \end{aligned}$$

where

$$\left.\begin{array}{llll} 0 \le c \le & o+j+k & 0 \le d \le & 2l-o-j-k \\ 0 \le c \le & l & 0 \le j \le & l \\ 0 \le u \le & l-j & 0 \le v \le & j \\ 0 \le s \le & l-k & 0 \le o \le & 2l \\ & & 1 \le k \le & l \\ & & 0 \le t \le & k \\ l = & c+d & l = & s+t+u+v \end{array}\right\} \qquad (7.2.3)$$

$o, j, k, s, t, u, v, c, d$ integer

We denote by $G(l)$ the set of all feasible solutions $x = (o, j, k, s, t, u, v, c, d)$ according to (7.2.3).

Next, we prove that g is bounded from below by $2l$. We start with summing up all terms in $l-k-s$ and all terms in $k-t$. This leads to

$$\begin{aligned} g(o,j,k,s,t,u,v,c,d) &= s\,c+(l-k-s)\,(2l+u+j+k) \\ &\quad +(2l-o)\,u+(2l-o)\,c \\ &\quad +u\,c+(l-j-u)\,c+(l-j-u)\,(o+j+k-c) \\ &+ t\,d+(k-t)\,(2l+v-j-k)+o\,v+o\,d+v\,d \\ &\quad +(j-v)\,d+(j-v)\,(2l-o-j-k-d). \end{aligned}$$

All terms of g remain non-negative. In the next step, we substitute t by $l-s-u-v$ and d by $l-c$. We achieve

$$\begin{aligned} g(o,j,k,s,t,u,v,c,d) &= s\,c+(l-k-s)\,(2l+u+j+k) \\ &\quad +(2l-o)\,u+(2l-o)\,c \\ &\quad +u\,c+(l-j-u)\,c+(l-j-u)\,(o+j+k-c) \\ &+ (l-s-u-v)\,(l-c)+(k-l+s+u+v)\,(2l+v-j-k) \\ &\quad +o\,v+o\,(l-c)+v\,(l-c) \\ &\quad +(j-v)\,(l-c)+(j-v)\,(2l-o-j-k-l+c). \end{aligned}$$

Case 2.1: If $l-k-s \geq 1$, g is bounded from below by $2l$ since all terms of g are non-negative.[1]

Case 2.2: Suppose that $l-k-s=0$. Then, $k+s=l$ where $k \geq 1$ and $s \leq l-1$.

$$\begin{aligned} g(o,j,k,s,t,u,v,c,d) &= (l-k)\,c+(2l-o)\,u+(2l-o)\,c \\ &\quad +u\,c+(l-j-u)\,c+(l-j-u)\,(o+j+k-c) \\ &+ (k-u-v)\,(l-c)+(u+v)\,(2l+v-j-k) \\ &\quad +o\,v+o\,(l-c)+v\,(l-c) \\ &\quad +(j-v)\,(l-c)+(j-v)\,(2l-o-j-k-l+c) \end{aligned}$$

g can be written as the sum of a function which depends on u and v and a function which is independent of u and v:

$$g(o,j,k,s,t,u,v,c,d) = \hat{g}(o,j,k,c)+\bar{g}(o,j,k,u,v,c)$$

[1] $g(o,j,k,s,t,u,v,c,d) \geq (l-k-s-1)\cdot(2l+u+j+k)+2l+u+j+k \geq 2l+1$ since $k \geq 1$.

where

$$\begin{aligned}
\hat{g}(o,j,k,c) &= (l-k)\,c + (2l-o)\,c + (l-j)\,c + (o+j+k-c)\,(l-j) \\
&\quad +k\,(l-c) + o\,(l-c) + j\,(l-c) + j\,(2l-o-j-k-l+c) \\
&= (l-k)\,c + (2l-o)\,c + (o+j+k)\,(l-j) + k\,(l-c) \\
&\quad +o\,(l-c) + j\,(2l-o-j-k) \\
\bar{g}(o,j,k,u,v,c) &= (2l-o)\,u + u\,c + (-u)\,c + (o+j+k-c)(-u) \\
&\quad (-u-v)\,(l-c) + (u+v)\,(2l+v-j-k) + o\,v \\
&\quad +v\,(l-c) + (-v)(l-c) + (-v)\,(2l-o-j-k-l+c) \\
&= u\,(3l-2o-2j-2k+2c) + u\,v + v^2 + v\,2o
\end{aligned}$$

$\bar{g}(o,j,k,u,v,c) \geq 0$ on $G(l)$, since $0 \leq d \leq 2l-o-j-k$ and $d = l-c$ implying that

$$c \geq -l + o + j + k.$$

Next, we show that $\hat{g}(o,j,k,c) \geq 2l$.

$$\begin{aligned}
\hat{g}(o,j,k,c) &= (l-k)\,c + (2l-o)\,c + (o+j+k)\,(l-j) \\
&+ \quad k\,(l-c) + o\,(l-c) + j\,(2l-o-j-k)
\end{aligned}$$

Since $k \geq 1$, at least one tram of type τ_3 must be assigned to stack 1. This implies that $2l - o \geq 1$. We achieve that

$$\begin{aligned}
\hat{g}(o,j,k,c) &= (l-k)\,c + (2l-o-1)\,c + (o+j+k-1)\,(l-j) \\
&+ \quad (k-1)\,(l-c) + o\,(l-c) + j\,(2l-o-j-k) + 2l - j - c + c
\end{aligned}$$

Case 2.2.1: If $j = 0$, then $\hat{g}$ is bounded from below by $2l$.

Case 2.2.2: If $j > 0$, then

$$\begin{aligned}
\hat{g}(o,j,k,c) &= (l-k)\,c + (2l-o-1)\,c + (o+j+k-1)\,(l-j) \\
&+ \quad (k-1)\,(l-c) + o\,(l-c) + j\,(2l-o-j-k) + 2l - j
\end{aligned}$$

Case 2.2.2.1: If $2l - o - j - k \geq 1$, then $\hat{g}$ is bounded from below by $2l$.

Case 2.2.2.2: We assume that $2l = o + j + k$. By $c \geq -l + o + j + k$, we achieve that $c = l$.

$$\begin{aligned}
\hat{g}(o,j,k,c) &= (l-k)\,c + (2l-o-1)\,c + (o+j+k-1)\,(l-j) \\
&+ \quad (k-1)\,(l-c) + o\,(l-c) + 2l - j \\
&= (l-k)\,l + (2l-o-1)\,l + (2l-1)(l-j) + 2l - j
\end{aligned}$$

Since $j \geq 1$ and $k \geq 1$, at least two trams of type τ_3 must be assigned to stack 1. This implies that $2l - o \geq 2$.

$$\begin{aligned} \hat{g}(o, j, k, c) &= (l-k)\, l + (2l - o - 1)\, l + (2l-1)(l-j) + 2l - j \\ &= (l-k)\, l + (2l - o - 2)\, l + (2l-1)(l-j) + 2l + l - j \end{aligned}$$

As a consequence, $\hat{g}$ is bounded from below by $2l$ because $l \geq j$.

Summarizing the discussion, we obtain that, in all cases, $\hat{g}$ is bounded from below by $2l$ and $\bar{g}$ by 0. Hence, $2l$ is a lower bound for g.

Finally, we have to examine how many shunting movements an optimal offline algorithm OPT needs in order to serve $\mathcal{A}_2$. OPT assigns the trams of type τ_2 to the positions in stack 2. The other trams are assigned to stack 1. All the trams can leave the depot in the opposite order of their arrival without shunting. Therefore, OPT yields an assignment without any shunting movement.

In conclusion, the (arbitrary) online algorithm ALG needs at least $\min\{l^2, 2l\} = O(N)$ shunting movements to serve the arrival sequence $\mathcal{A}_1$ or $\mathcal{A}_2$ for which an optimal offline algorithm determines a solution without shunting. ❑

7.2.2 Performance on $\mathcal{I}_1$

Analogously to Theorem 7.2.4, we show that every deterministic online algorithms yields a solution with a number of shunting movements that is linear in the number of trams where an optimal offline algorithms only requires one shunting movement.

Theorem 7.2.5: Every deterministic online algorithm yields a solution of at least $(\frac{N-1}{2})$ shunting movements for a class of instances in $\mathcal{I}_1$ where N denotes the number of arriving trams, N odd.

Proof: We prove the theorem by using a similar argumentation as in the proof of Theorem 7.2.3. We will construct two arrival sequences which are to be served by an arbitrary online algorithm. The decision which sequence is chosen depends on the assignment of the first arriving tram. We show that for these instances any arbitrary online algorithm needs at least $\frac{N-1}{2}$ shunting movements whereas an optimal offline algorithm needs at most one shunting movement.

First, we consider the following two instances. We construct instances with two stacks and N departures. The stacks consists of N positions. N is chosen to be odd, i.e., $N = 2l + 1$, $N \geq 5$. The first stack is of length $l + 1$ and the second has length l, $l \geq 2$.

As departure sequence we choose $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ with the following types

$$t(d_i) = \begin{cases} \tau_1 & \text{if } 1 \leq i \leq l \\ \tau_2 & \text{if } l+2 \leq i \leq N \\ \tau_3 & \text{if } i = l+1 \end{cases} .$$

The first arrival sequence is $\mathcal{A}_1 = \{a_1^1, a_2^1, \ldots, a_N^1\}$ with the types

$$t(a_i^1) = \begin{cases} 1 & \text{if } \tau_1 \leq i \leq l \\ \tau_2 & \text{if } l+1 \leq i \leq N-2 \ \vee \ i = N \\ \tau_3 & \text{if } i = N-1 \end{cases} .$$

The types of the second arrival sequence $\mathcal{A}_2 = \{a_1^2, a_2^2, \ldots, a_N^2\}$ are defined as follows:

$$t(a_i^2) = \begin{cases} \tau_1 & \text{if } i = 1 \quad \vee \quad 3 \leq i \leq l+1 \\ \tau_2 & \text{if } l+2 \leq i \leq N \\ \tau_3 & \text{if } i = 2 \end{cases} .$$

Both arrival sequences share the property that the first arrival is of type τ_1.

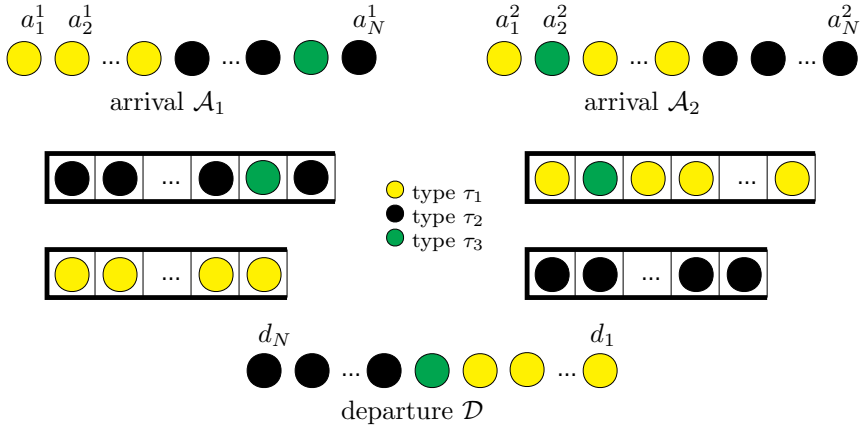

Figure 7.2.8: Optimal assignments determined by an optimal offline algorithm.

Figure 7.2.8 shows optimal assignments determined by an optimal offline algorithm. For both arrival sequences, an optimal offline algorithm yields assignments requiring one shunting movement.

In the following, we show that each online algorithm needs at least $l = \frac{N-1}{2}$ shunting movements by choosing arrival sequence $\mathcal{A}_1$ or $\mathcal{A}_2$ based on the first decision of the online algorithm.

In both cases, the first arrival given to the online algorithm is the arrival of a tram of type τ_1. The online algorithm has two possibilities to assign this tram (cf. Figure 7.2.9). Firstly, the online algorithm can assign this tram to the bottom position p_1 of stack 1. Secondly, the tram can be assigned to the bottom position p_{l+2} of stack 2. In the first case, the arrival sequence given to the online algorithm is $\mathcal{A}_1$. In the second case, $\mathcal{A}_2$ is given to the online algorithm.

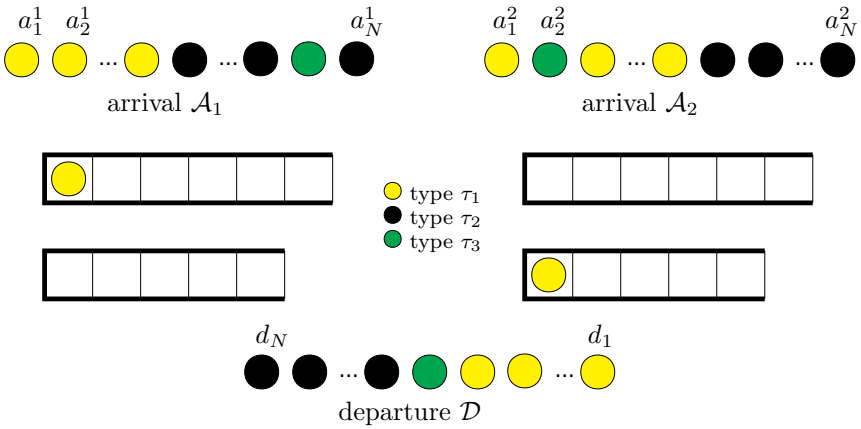

Figure 7.2.9: The first choice made by the online algorithm.

Next, we analyze which consequences the first assignment of the online algorithm has.

Case 1: In the arrival sequence $\mathcal{A}_1$, the next $l-1$ trams to be assigned by the online algorithm are of type τ_1. If all trams are assigned to stack 1, this stack is filled with l trams of type τ_1 except of the top position. To this position a tram of a different type has to be assigned. Hence, all l trams of type τ_1 which are supposed to leave as first are blocked by this tram. Consequently, l shunting movements are necessary for the departure.

Instead of assigning all trams of type τ_1 to stack 1, the online algorithm could have assigned at least one of them to stack 2. However, to the top positions p_{l+1} and p_N trams of type τ_2 or τ_3 have to be assigned. All trams of type τ_1 in stack 1 have to be shunted with the tram at position p_{l+1}. Additionally, all trams of type τ_1 in stack 2 have to be shunted with the tram at position p_N so that all l trams of type τ_1 have to be shunted.

In this case, at least l shunting movements are required by the assignment made by the online algorithm ALG.

Case 2: Next, we consider the case where the online algorithm assigns the first tram to stack 2. In this case, the algorithm has to serve arrival sequence $\mathcal{A}_2$. In this sequence, the next tram to be assigned is of type τ_3. The algorithm has two alternatives to assign this tram. Either it is assigned to the bottom position of stack 1 or it is assigned to the on-top position of tram a_1^2 (cf. Figure 7.2.10).

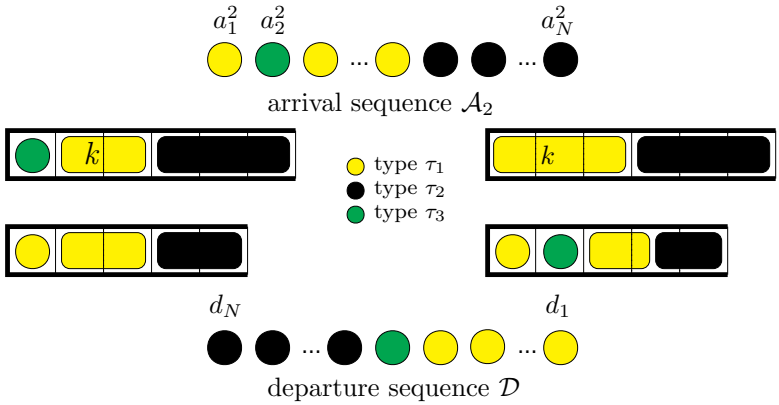

Figure 7.2.10: The possibilities of the online algorithm to serve $\mathcal{A}_2$.

Case 2a: Let us assume that the second tram is assigned to position p_1 of stack 1. The next $l-1$ trams in $\mathcal{A}_2$ are of type τ_1. If the online algorithm assigns these trams to stack 1, a tram of type τ_2 must be assigned to the top position of this stack. Then, all trams in stack 1 have to depart earlier than the tram of type τ_2 so that at least l shunting movements are required to satisfy the departures.

We denote by k the number of trams of type τ_1 that the online algorithm assigns to stack 1, $1 \leq k \leq l-1$. The remaining $l-k-1$ trams of type τ_1 are assigned to stack 2. Then, at least $(k+1)(l-k)$ shunting movements are necessary for stack 1, since all $k+1$ trams of type τ_1 and τ_3 have to leave the stack before the $l-k$ trams of type τ_2. For stack 2, $k(l-k)$ are necessary because k trams of type τ_2 are standing in front of $l-k$ trams of type τ_1. In total, at least l shunting movements are necessary.[2]

Case 2b: If the online algorithm assigns the second tram to stack 2, this single assignment leads to one shunting movement, because all trams of type τ_1 have to leave the stacks earlier than the tram of type τ_3. The next $l-1$ trams are of type τ_1. Since there are only $l-2$ positions left in stack 2, some trams of type τ_1 must be assigned to stack 1. Let us assume that k trams of type τ_1 are assigned to stack 1 and the remaining trams are assigned to stack 2, $1 \leq k \leq l-1$. In stack 1, all trams of type τ_1 have to leave before the all trams of type τ_2. In stack 2, all trams of type τ_1 and τ_3 are leaving before the trams of type τ_2. Additionally, the tram of type τ_1 at the bottom and the tram of type τ_3 have to be shunted. Hence, at least $k(l+1-k)$ shunting movements are required for stack 1. For stack 2, $(k-1)(l-k+1)+1$ for stack 2 shunting movements are required, because $k-1$ trams of type τ_2 have to shunted with $l-k$ trams of type τ_1 and of tram of type τ_1 has to be shunted with the tram of type τ_3. Consequently, the online

[2] $(k+1)(l-k)+k(l-k) = k(l-k)+l-k+k+k(l-k-1) \geq l$ since $1 \leq k \leq l-1$

algorithm needs more than l shunting movements for this instance.[3]

Summarizing the discussion, by its first decision we force the online algorithm to yield an assignment requiring at least l shunting movements for instances for which the optimal offline algorithm yields a solution with one shunting movement. ❑

Theorem 7.2.4 and Theorem 7.2.5 imply that

Corollary 7.2.6: Every deterministic online algorithm for TDP is not (c, d)-competitive if $c < \frac{N-1}{2}$ or $d < \frac{N}{3}$.

Theorem 7.2.6 implies that every online algorithm ALG has competitive ratios c and d that are at least linear in the number of arrivals. A trivial upper bound on the competitive ratios of an online algorithm ALG solving the tram dispatch problem is given by $\frac{1}{2}\ N(N-1)$, i.e., the maximum number of shunting movements that an algorithm needs for N trams and one stack. In the correspondent worst-case instance for this bound, ALG assigns the trams in such a way that the trams have to be shunted pairwise without any exception. Assuming that the depot consists of only one stack and all trams are of different type, we obtain $\frac{1}{2}\ N(N-1)$ shunting movements. For one stack instances, the following online algorithm is optimal. This online algorithm assigns the trams to the stack positions in the order of arrival and to the departures following the last-in-first-out principle. The maximum number of shunting movements for two stack instances is given by

$$\frac{1}{2}\ P_1(P_1-1) + \frac{1}{2}\ P_2(P_2-1)$$

where P_r denotes the number of positions in stack r, $r \in \{1,2\}$ and $N = P_1 + P_2$. This number is equal to

$$\frac{1}{2}\ N^2 - (N-P_1)P_1 - \frac{1}{2}\ N$$

which is maximal for $P_1 = 1$ (or $P_1 = N-1$). Similar upper bounds can be determined analogously for instances with an arbitrary number of stacks by computing the maximal value of

$$\frac{1}{2}\ \sum_{r=1}^{R} P_r\ (P_r - 1) \quad \text{where} \quad \sum_{r=1}^{R} P_r = N$$

However, the upper bound of $\frac{1}{2}\ N(N-1)$ holds for every arbitrary number of stacks.

[3] $k(l+1-k)+(k-1)(l-k+1)+1 = (k-1)(l+1-k)+l+1-k+k-1+(k-1)(l-k)+1 \geq l+1$ for $1 \leq k \leq l-1$

7.2.3 Randomization

If we consider the proofs of Theorem 7.2.4 and Theorem 7.2.5 more intensively, we observe that the arbitrarily chosen online algorithm ALG is forced to yield a bad solution by its first decision. For such two stack instances, a randomized online algorithm RAND would have chosen to assign the first tram to stack 1 with a fixed probability p and to stack 2 with a probability of $1-p$. The probability p can be regarded as being known to the cruel (oblivious) adversary constructing the arrival sequence.

Depending on p, the oblivious adversary always chooses the arrival sequence which will force RAND to yield a solution with higher cost. If $p \geq \frac{1}{2}$ then $\mathcal{A}_1$ is given to RAND. Otherwise, RAND has to serve $\mathcal{A}_2$. As we have observed in the proofs of Theorem 7.2.4 and Theorem 7.2.5, all further decision of ALG and RAND will not help to improve the performance. Consequently, against an oblivious adversary the expected number of shunting movements for RAND is at least $\frac{1}{2} \cdot \frac{N-1}{2}$ for instances of $\mathcal{I}_1$ and at least $\frac{1}{2} \cdot \frac{N}{3}$ shunting movements for instances of $\mathcal{I}_0$. We achieve that

Corollary 7.2.7: Every randomized online algorithm for TDP is not (c,d)-competitive against any oblivious adversary where $c < \frac{N-1}{4}$ and $d < \frac{N}{6}$.

Adaptive adversaries are able to react on the action of RAND in such a way that they are able to wait with constructing $\mathcal{A}$ until RAND has made its first decision. Hence, randomization does not help to improve the lower bound on the competitiveness if we are faced with adaptive adversaries.

7.2.4 The Type Mismatch Problem

For the worst-case instances Theorem 7.2.4 and Theorem 7.2.5, we observe the following performance of online algorithms for TMP.

Theorem 7.2.8: Any arbitrary online algorithm for TMP needs at least two type mismatches for a class of instances of $\mathcal{I}_0$.

Proof: We show that for the considered class of instances of $\mathcal{I}_0$ of Theorem 7.2.4, any arbitrary online algorithm ALG for TMP needs at least two type mismatches. Depending on the first decision of ALG, a cruel adversary decides whether arrival sequence $\mathcal{A}_1$ or $\mathcal{A}_2$ is given to ALG. If ALG assigns the first tram of type τ_1 to stack 1, then $\mathcal{A}_1$ has to be served. Otherwise, $\mathcal{A}_2$ is given to ALG.

Recall that $\mathcal{A}_1$, $\mathcal{A}_2$, and the departure sequence $\mathcal{D}$ are given by the following sequence of tram types beginning with a_1^1, a_1^2, and d_1, respectively (see also Figure 7.2.5).

$$\begin{aligned}
\mathcal{A}_1 &: (\tau_1, \ldots, \tau_1, \tau_1, \ldots, \tau_1, \tau_2, \ldots, \tau_2, \tau_3, \ldots, \tau_3, \tau_3, \ldots, \tau_3, \tau_2, \ldots, \tau_2) \\
\mathcal{A}_2 &: (\tau_1, \ldots, \tau_1, \tau_3, \ldots, \tau_3, \tau_3, \ldots, \tau_3, \tau_1, \ldots, \tau_1, \tau_2, \ldots, \tau_2, \tau_2, \ldots, \tau_2) \\
\mathcal{D} &: (\tau_1, \ldots, \tau_1, \tau_2, \ldots, \tau_2, \tau_3, \ldots, \tau_3, \tau_3, \ldots, \tau_3, \tau_1, \ldots, \tau_1, \tau_2, \ldots, \tau_2)
\end{aligned}$$

Each subsequence $\tau_i, \ldots, \tau_i$ consists of l trams of departures of type τ_i, $1 \leq i \leq 3$, $l \geq 2$. Hence, there are $N = 6l$ trams and departures. The first stack contains $4l$ and the second stack $2l$ positions.

If ALG assigns the first tram of type τ_1 to the first stack, the next $2l-1$ trams are of type τ_1. In any case, after assigning all the trams to the stack positions, the top-most trams in both stack have a type different than τ_1. Consequently, we need at least one type mismatch for the first departure. Since throughout the whole thesis, we assume that there are as many trams of each type as there departures of the same type, ALG needs at least two type mismatches if it decides to assign the first tram to stack 1.

In the case that ALG assigns the first tram to stack 2, $\mathcal{A}_2$ is given to ALG. The next $l-1$ trams are of type τ_1 followed by $2l$ trams of type τ_3. Then, l trams of type τ_1 arrive at the depot. The $2l$ trams of type τ_2 arrive at the depot as the last. Consequently, these trams are assigned to the top-most positions. Since there are only at most $2l-1$ positions in stack 2 to which no tram of type τ_1 and type τ_3 have been assigned, a tram of type τ_2 is assigned to the top position of stack 1.

If there is a tram of type τ_2 assigned to the top position of stack 2, then one type mismatch is necessary for the first departure resulting in at least two type mismatches. Otherwise, all trams of type τ_2 are assigned to the $2l$ top-most positions in stack 1. In this case, we need l type mismatches for last l departures of type τ_2. ❑

The lower bound of two type mismatches for serving $\mathcal{A}_1$ is tight if the first tram is assigned to stack 1. ALG may assign all the trams of type τ_1, except of the first one, to stack 2. On top of these $2l-1$ trams a tram of type τ_2 is assigned. The remaining trams of type τ_2 and τ_3 are assigned to stack 1 in the order of their arrival. For this assignment, ALG needs only two type mismatches: One for the first departure which is served by the tram of type τ_2 at the top position of stack 2 and one for the last departure which is served by tram a_1^1 at the bottom position of stack 1.

The best assignments that an online algorithm can achieve are as follows. We give the assignments by the tram types for each stack beginning with the bottom positions:

$$\begin{aligned}
\mathcal{A}_1: &\quad \text{stack 1 } (\tau_1, \tau_2, \ldots, \tau_2, \tau_3, \ldots, \tau_3, \tau_3, \ldots, \tau_3, \tau_2, \ldots, \tau_2)\\
&\quad \text{stack 2 } (\tau_1, \ldots, \tau_1, \tau_1, \ldots, \tau_1, \tau_2)\\
\mathcal{A}_2: &\quad \text{stack 1 } (\tau_1, \ldots, \tau_1, \tau_3, \ldots, \tau_3, \tau_3, \ldots, \tau_3, \tau_1, \ldots, \tau_1, \tau_2)\\
&\quad \text{stack 2 } (\tau_1, \tau_2, \ldots, \tau_2, \tau_2, \ldots, \tau_2)
\end{aligned}$$

The above assignment of trams to stack positions for $\mathcal{A}_2$ requires four type mismatches: one for the first departure, for the last departure, one for departure

d_{2l+1} (the first one of type τ_3), and one for departure d_{4l+1} (the forst of of the second block of departures of type τ_1).

If the first arriving tram is assigned to stack 2, then any arbitrary online algorithm needs at least four type mismatches for serving $\mathcal{A}_2$. All the trams of type τ_2 are assigned as the last. Any other assignment than the one presented above requires at least four type mismatches. In these assignments, at least two trams of type τ_2 are assigned to stack 1 and at least at the two bottom positions of stack 2 there are trams of a type different than τ_2. Note that the last l departures are of type τ_2.

For the class of instances of Theorem 7.2.5, there is an online algorithm yielding a solution that requires only two type mismatches, i.e., the same number of type mismatches that an optimal offline algorithm needs at least for instances of $\mathcal{I}_1$. Hence, by considering the classes of instances of Theorem 7.2.4 and Theorem 7.2.5, we can state the following result:

Corollary 7.2.9: There is no deterministic online algorithm for TMP which is (c, d)-competitive where $d < 2$.

The trivial upper bounds on c and d are given by the worst-case performance for any arbitrary algorithm. For all instances, at most N type mismatches are required for N departures. Consequently, an upper on c is given by $\frac{N}{2}$ and an upper bound on d is given by N.

7.3 The Departure Problem

In this section, we restrict ourselves to the online versions of DTDP and DTMP. We assume that we are given an assignment π_X of trams to stack positions. The departure sequence $\mathcal{D}$ is given departure by departure to an online algorithm ALG. Before the next departure is revealed, ALG has to assign to the actual departure a tram stored in some stack. Depending on the problem (DTDP or DTMP), ALG has to choose a tram so that this assignment is type-preserving or shunting-free. According to Definition 7.1.1, we examine separately the case for which an optimal offline algorithm OPT yields a (shunting-free and type-preserving) solution with zero cost and the case for which OPT determines a solution with non-zero cost. We start with considering the online DTDP.

7.3.1 Online DTDP

By $\mathcal{I}_0$, we denote the class of instances of DTDP for which an optimal offline algorithm OPT yields a solution not requiring shunting. Hence, $\mathcal{I}_0$ contains all instances having a shunting-free and type-preserving solution.

Theorem 7.3.10: For every deterministic online algorithm ALG, there are instances $I_N \in \mathcal{I}_0$ for which $cost_{\mathrm{ALG}}(I_N) \geq \frac{N}{4}$ where N denotes the number of trams in I_N.

Proof: We prove the theorem by giving an instance of $\mathcal{I}_0$ with N trams for which any arbitrary deterministic online algorithm yields a solution requiring at least $\frac{N}{4}$ shunting movements.

We restrict ourselves to such online algorithms which in each step try to avoid shunting by choosing in some stack the top-most tram of suitable type. Any other online algorithm choosing a tram at a lower position in the same stack yields a solution of higher cost, i.e., requiring more shunting movements.

We consider an instance of $N = 4l$ trams of three types τ_1, τ_2, and τ_3 stored in two stacks of length $2l$, $l \in \mathbb{N}$.

The stacks are given by the following type sequences starting with the bottom positions:

$$\begin{array}{ll} \text{stack 1:} & (\tau_2, \tau_1, \tau_2, \tau_1, \ldots, \tau_2, \tau_1) \\ \text{stack 2:} & (\tau_3, \tau_1, \tau_3, \tau_1, \ldots, \tau_3, \tau_1) \end{array}$$

The departure sequence $\mathcal{D}$ is divided into l subsequences consisting of four departures. Each subsequence is either $(\tau_1, \tau_2, \tau_1, \tau_3)$ or $(\tau_1, \tau_3, \tau_1, \tau_2)$. Each subsequence corresponds to a phase in which the cruel adversary (cf. page 123) proceeds with constructing $\mathcal{D}$ by choosing the subsequences depending on the actions of an arbitrary online algorithm ALG.

It is obvious that OPT finds a shunting-free and type-preserving assignment for a departure sequence containing the above subsequences.

In both cases, the adversary chooses as first a departure of type τ_1. ALG has two possibilities to choose a tram of type τ_1. Either ALG chooses the tram from the top position of stack 1 or it chooses the tram from the top position of stack 2 (see also Figure 7.3.11). In the first case, the adversary chooses the second subsequence. In the other case, the first subsequence is given to ALG.

ALG needs at least one shunting movement to serve the departures of each subsequence if we restrict ourselves to such online algorithms which always choose the top-most tram of suitable type for each stack.

After the four departures of a phase are served, a new phase begins (cf. Figure 7.3.11). The stacks are similar to the initial stacks.

By repetition, ALG derives an assignment that causes exactly one shunting movement per phase resulting in a total number of at least $\frac{N}{4}$ shunting movements. ❑

For instances in $\mathcal{I}_0$, another lower bound on the competitive ratio is obtained by analyzing the performance of an arbitrarily chosen online algorithm ALG for another subclass of instances of $\mathcal{I}_0$.

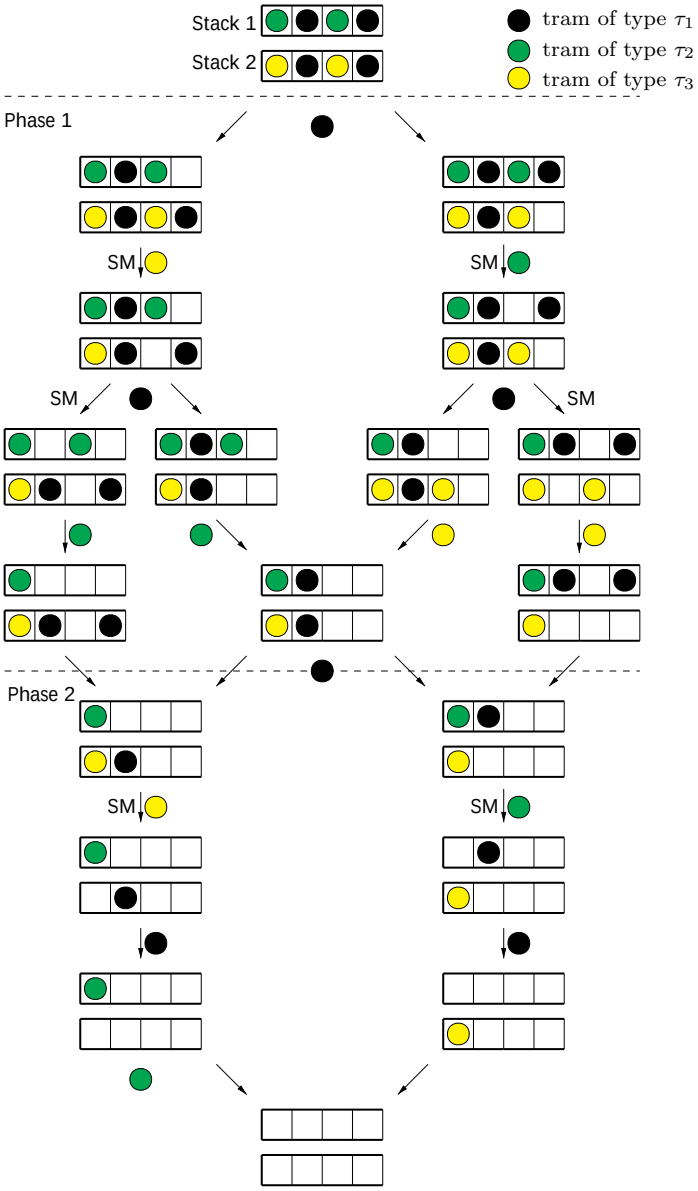

Figure 7.3.11: The instance for $l = 2$ for which the solution obtained by ALG requires at least l shunting movements (denoted by SM) whereas OPT yields a shunting-free solution.

Theorem 7.3.11: Every deterministic online algorithm needs at least $\frac{N}{2} - \frac{R}{2}$ shunting movements for a subclass of instances in $\mathcal{I}_0$ where N denotes the number of trams and R denotes the number of stacks.

Proof: We consider R stacks of length l and $N = l \cdot R$ trams of $N - R + 1$

types, $R > 1$, R even, and $l > 1$. The r-th stack contains trams of the following types beginning with the bottom element ($1 \leq r \leq R$):

$$(\tau_{r(l-1)}, \tau_{r(l-1)-1}, \ldots, \tau_{r(l-1)-l+2}, \tau_0)$$

The top element of each stack is of type τ_0. The remaining trams are of pairwise different type (cf. Figure 7.3.12).

We divide the construction of the departure sequence $\mathcal{D}$ into R phases (cf. Figure 7.3.12).

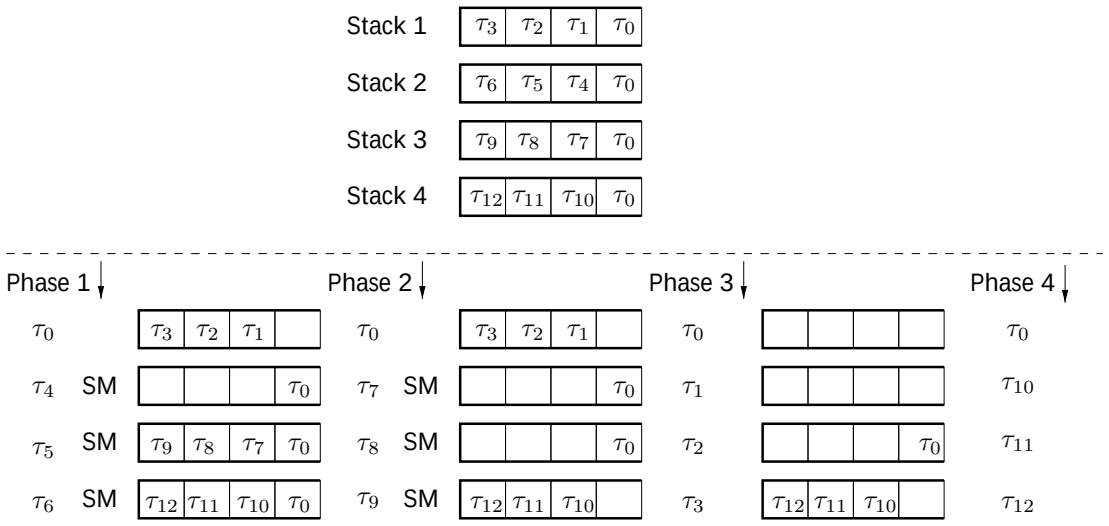

Figure 7.3.12: The instance for $R = 4$, $N = 16$, and $l = 4$ for which the solution obtained by an online algorithm ALG requires at least $\frac{N}{2} - \frac{R}{2} = 6$ shunting movements (denoted by SM) whereas OPT yields a shunting-free solution.

At the beginning of the first phase, the cruel adversary gives a departure of type τ_0. ALG has to choose a tram of this type from the top of one stack. Next, the adversary continues with the $l-1$ departures fitting to the types of a different stack. OPT is able to assign these trams without shunting. At the beginning of the first phase, OPT chooses the top element (of type τ_0) of this stack, i.e., of the stack containing the trams fitting to the next $l-1$ departures.

Since ALG has chosen the "wrong" tram, in this phase ALG needs $l-1$ shunting movements for this part of $\mathcal{D}$. After all departures are assigned, the next phase starts with a departure of type τ_0 again.

Once again, ALG has to assign a tram of type τ_0 from the top of one stack. ALG has two possibilities to choose this tram. First, ALG can choose this tram from the top of a stack that has been almost emptied except of the tram at the

top. Secondly, ALG can decide to take a tram standing at the top position of a *complete stack*, i.e., a stack from which no tram has been assigned by ALG until now.

In both cases, the cruel adversary chooses departures corresponding to another complete stack. As in phase one, ALG needs $l-1$ shunting movements for the departures of this phase whereas OPT avoids shunting.

The cruel adversary can proceed with this strategy at least until phase $\frac{R}{2}+1$. ALG fails in at least $\frac{R}{2}$ phases which results in $\frac{R}{2}(l-1)$ shunting movements. In every phase, OPT is able to assign the trams of one stack to the departures of this phase without shunting. Consequently, the constructed instance belongs to $\mathcal{I}_0$.

The cruel adversary forces ALG to obtain a solution that requires at least $\frac{R}{2}\cdot(\frac{N}{R}-1)$ shunting movements whereas OPT yields an assignment without shunting. For this class of instances, we achieve that for the instance I_N constructed above

$$cost_{\mathrm{ALG}}(I_N) \geq \frac{R}{2}\cdot\left(\frac{N}{R}-1\right) = \frac{N}{2}-\frac{R}{2} \tag{7.3.4}$$

For two stacks, we achieve that $cost_{\mathrm{ALG}}(I_N) \geq \frac{N}{2}-1$. ❑

In the next step, we examine instances of DTDP for which OPT needs at least one shunting movement. Analogously to Theorem 7.3.11, we obtain the following result:

Theorem 7.3.12: Every deterministic online algorithm needs at least $\frac{N}{2}-1$ shunting movements for instances in $\mathcal{I}\backslash\mathcal{I}_0$ where N denotes the number of trams.

Proof: We consider a two stack instance. Each stack contains $\frac{N}{2}$ trams, where N is even. For each stack, the tram types are given by the following type sequence starting with the bottom position:

$$\begin{array}{ll} \text{stack 1} & (\tau_3,\tau_3,\ldots,\tau_3,\tau_1,\tau_3) \\ \text{stack 2} & (\tau_2,\tau_2,\ldots,\tau_2,\tau_1,\tau_2) \end{array}$$

We start constructing the departure sequence $\mathcal{D}$ by giving a departure of type τ_1.

The arbitrarily chosen online algorithm ALG has two possibilities to assign a tram of type τ_1 to the first departure. First, it can assign the tram of type τ_1 stored in stack 1. Secondly, it may choose the tram of type τ_1 stored in stack 2. In both cases, ALG needs one shunting movement for this departure.

If ALG chooses the tram stored in stack 1, then we complete $\mathcal{D}$ in the following way (starting with d_2):

$$(\tau_2,\tau_2,\ldots,\tau_2,\tau_2,\tau_3,\tau_1,\tau_3,\ldots,\tau_3,\tau_3)$$

In the second case, we choose

$$(\tau_3, \tau_3, \ldots, \tau_3, \tau_3, \tau_2, \tau_1, \tau_2, \ldots, \tau_2, \tau_2)$$

In both cases, ALG needs $\frac{N}{2} - 1$ shunting movements for $\mathcal{D}$ (including the first shunting movement). In contrast, OPT needs only one shunting movement (the one for the first departure). ❑

As a consequence of Theorem 7.3.11 and Theorem 7.3.12, we obtain the following corollary.

Corollary 7.3.13: No deterministic online algorithm for DTDP is (c, d) – competitive where $c < \frac{N}{2} - 1$ or $d < \frac{N}{2} - 1$.

Randomization

For (c, d)-competitive deterministic online algorithms, Corollary 7.3.13 establishes a lower bound of $\frac{N}{2} - 1$ on c as well as on d. In the corresponding proofs, we considered two stack instances for which any deterministic online algorithm was forced to fail by its first decision.

For these instances, a randomized online algorithm RAND behaves as follows. RAND decides between the two alternatives for the first decision where p denotes the probability to assign a tram of stack 1 and $1 - p$ denotes the probability to assign a tram of stack 2. Depending on p, the oblivious adversary chooses $\mathcal{D}$. If $p \geq \frac{1}{2}$, then the oblivious adversary proceeds in the same as the cruel adversary in the case that ALG has chosen for the first departure a tram of stack 1. If $p < \frac{1}{2}$, then the oblivious adversary behaves in the same as the cruel adversary in the case that ALG has chosen for the first departure a tram of stack 2. Hence, the expected cost of RAND are at least half of the cost of ALG so that we obtain the following result:

Theorem 7.3.14: No randomized online algorithm is (c, d)–competitive against any oblivious adversary where $c < \frac{N}{4} - \frac{1}{2}$ or $d < \frac{N}{4} - \frac{1}{2}$.

Against adaptive (online and offline) adversaries, no improvement is possible because every online algorithm fails by its first decision.

7.3.2 Online Algorithms for DTDP

Next, we derive an upper bound on the performance for a deterministic online algorithm, called **GREEDY-DTDP**. GREEDY-DTDP assigns to each departure of $\mathcal{D}$ a top-most tram of suitable type. If two (or more) trams in different stacks satisfy the property of being a top-most tram of the required type, then GREEDY-DTDP chooses a tram in a fixed, deterministic way. For instance,

GREEDY-DTDP chooses the tram that is stored in the stack having the lowest stack number among the corresponding stacks.

Proposition 7.3.15: GREEDY-DTDP is an optimal algorithm for arbitrary instances consisting of one stack only. For instances consisting of trams of pairwise distinct types, every algorithm computing a type-preserving assignment is optimal.

Proof: For one stack instances, GREEDY-DTDP proceeds in the same way as OPT, since OPT always chooses the top-most tram for one-stack instances. If all trams have distinct types, there is only one type-preserving assignment. ❑

For GREEDY-DTDP, we observe the following lower bounds on the competitive ratios c and d.

Theorem 7.3.16: GREEDY-TDTP is not (c, d) – competitive where $c < N-3$ or $d < N-2$. (N denotes the number of trams.)

Proof: First, we give a class of instances I_N of $\mathcal{I}_0$ for which GREEDY-DTDP needs $N-2$ shunting movements. Then, we introduce a class of instances of $\mathcal{I}\backslash\mathcal{I}_0$ for which GREEDY-DTDP yields a solution of at least $N-3$ shunting movements whereas an optimal offline algorithm needs exactly one shunting movement.

We construct the instance I_N as follows. I_N consists of two stacks where the first stack has only one position at which a tram of type τ_1 is stored. Stack 2 having $N-1$ positions is given by the following sequence of tram types beginning with the bottom position:

$$(\tau_2, \tau_2, \ldots, \tau_2, \tau_1)$$

The types of the departure sequence $\mathcal{D}$ are given as follows (starting with d_1):

$$(\tau_1, \tau_2, \tau_2, \ldots, \tau_2, \tau_1)$$

It it obvious that an optimal offline algorithm starts with assigning the trams of stack 2. The tram of stack 1 is assigned as last. This is possible without shunting. In contrast, GREEDY-DTDP starts with the tram of stack 1 resulting in $N-2$ shunting movements for the trams of type τ_2.

The instance I'_N belonging to $\mathcal{I}\backslash\mathcal{I}_0$ is also a two stack instance similar to I_N. The first stack has only one position at which a tram of type τ_1 is stored. Stack 2 differs in the two top-most positions in the way that the tram types of the corresponding trams are switched. On top of stack 2, there is a tram of type τ_2 followed by a tram of type τ_1 and $N-3$ remaining trams of type τ_2.

$$(\tau_2, \tau_2, \ldots, \tau_1, \tau_2)$$

The departure sequence $\mathcal{D}$ is the same as in I_N.

Once again, GREEDY-DTDP starts with the tram of stack 1 resulting in $N-3$ shunting movements. An optimal offline algorithm OPT starts with one shunting movement for the assignment of the tram of type τ_1 in stack 2. For the remaining departures, OPT does not require any shunting movements. ❑

Tighter bounds for the competitive ratios of GREEDY-DTDP are achieved in the following theorem.

Theorem 7.3.17: GREEDY-DTDP requires at least d shunting movements for a class of instances of $\mathcal{I}_0$ and at least d shunting movements for a class of instances of $\mathcal{I} \setminus \mathcal{I}_0$ where

$$d = \frac{1}{6}\, N^2 - \frac{1}{2}\, N + \frac{1}{3}$$

and N denotes the number of trams.

Proof: We prove the theorem by constructing a class of instances of $\mathcal{I}_0$ and a (related) class of instances of $\mathcal{I} \setminus \mathcal{I}_0$. An instance of these classes consists of $N = 2l$ trams of l types where $l = 2^i$ and $i \geq 2$ and $R = i + 1$ stacks.

For both classes, the type of each departure $d \in \mathcal{D}$ is given by the following sequence of types beginning with d_1:

$$(\tau_1, \tau_2, \tau_3, \ldots, \tau_{l-1}, \tau_1, \tau_l, \tau_{l-1}, \ldots, \tau_3, \tau_2, \tau_1).$$

We start with considering an instance of $\mathcal{I}_0$. For large N, we give the trams stored in the $R = i+1$ stacks by the following type sequences beginning with the bottom positions.

The first stack is given by

$$(\tau_1, \tau_1).$$

The second stack contains two trams of the following types

$$(\tau_2, \tau_3).$$

The third stack consists of

$$(\tau_4, \tau_5, \tau_6, \tau_7).$$

In stack j, $2 \leq j \leq R-1$, there are 2^{j-1} trams given by

$$(\tau_{2^{j-1}}, \tau_{2^{j-1}+1}, \ldots, \tau_{2^j-1}).$$

The last stack contains exactly l trams of pairwise different type:

$$(\tau_l, \tau_{l-1}, \ldots, \tau_3, \tau_2, \tau_1).$$

OPT yields a shunting-free solution in the following way. OPT starts with assigning the first $l-1$ departures to the last stack. Next, OPT assigns d_l to the top position of the first stack. Departure d_{l+1} is assigned to the tram at the bottom position of the last stack. The following $l-2$ departures are assigned the stack $R-1$, $R-2$, ... , 2. The last departure d_N is assigned to the tram at the bottom position of stack 1.

By definition, GREEDY-DTDP always chooses a tram of suitable type stored at the top-most position in the stacks. If there are more than one such trams, then GREEDY-DTDP chooses the tram from the stack having the smallest index. GREEDY-DTDP starts with assigning the first departure to stack 1. Departure d_2 and d_3 are assigned to stack 2 resulting in one shunting movement. The departures $d_4, \ldots, d_7$ are assigned to stack 3. For this assignment, six shunting movements are required. The next eight departure are assigned to stack 4 requiring 28 shunting movements. Departure d_l is assigned to the tram at the bottom position of the first stack. The last l departures are assigned to the last stack resulting in $\frac{l}{2}(l-1)$ shunting movements for this stack.

Summarizing, GREEDY-DTDP needs exactly $\frac{1}{2}\,2^{j-1}(2^{j-1}-1)$ shunting movements for stack j, $2 \le j \le i$, and $\frac{l}{2}(l-1)$ shunting movements for the last stack. Since $l = 2^i$ and $N = 2l$, this number of shunting movements is equal to

$$\begin{aligned}
\sum_{j=1}^{i-1} \frac{1}{2} 2^j (2^j - 1) + \frac{l}{2}(l-1) &= \sum_{j=1}^{i-1} 2^{2j-1} - \sum_{j=1}^{i-1} 2^{j-1} + \frac{l}{2}(l-1) \\
&= 2 \sum_{j=0}^{i-2} 4^j - \sum_{j=0}^{i-2} 2^j + \frac{l}{2}(l-1) \\
&= 2\,\frac{1}{3}\,(4^{i-1}-1) - 2^{i-1} + 1 + \frac{l}{2}(l-1) \\
&= \frac{2}{3}\left(\frac{l^2}{4} - 1\right) - \frac{l}{2} + 1 + \frac{l^2}{2} - \frac{l}{2} \\
&= \frac{2}{3}\,l^2 - l + \frac{1}{3} \\
&= \frac{2}{3}\,\frac{N^2}{4} - \frac{N}{2} + \frac{1}{3}.
\end{aligned}$$

The class of instances of $\mathcal{I} \setminus \mathcal{I}_0$ differs from the above class only in the first two stacks. Stack 1 is given by

$$(\tau_3, \tau_1)$$

and stack 2 is given by

$$(\tau_2, \tau_1).$$

The remaining $R-2$ stacks are given as above.

For this instance, GREEDY-DTDP yields an assignment of the same number of shunting movements. OPT needs one shunting movement either for departure d_{N-1} of type τ_2 or for departure d_{N-2} of type τ_3. ❑

In Theorem 7.3.17, we constructed instances for which OPT needs at most one shunting movement whereas GREEDY-DTDP behaves poorly. Given the last stack, we have constructed the remaining $R-1$ stacks in such a way that OPT succeeds and GREEDY-DTDP has to shunt all the trams in the stacks $2, \ldots, R$. We state the following conjecture.

Conjecture 7.3.18: GREEDY-DTDP is (c,d)–competitive where $c = d = \frac{1}{6}\,N^2 + O(N)$.

7.3.3 Online DTMP

Next, we consider the online DTMP. In this problem, we are given an assignment π_X of trams to stack positions. The departure sequence $\mathcal{D}$ is presented to ALG departure by departure. Before the next departure is revealed, ALG has to assign a tram at some position $p \in \mathcal{P}$ to the actual departure $d_j \in \mathcal{D}$. The assignment has to be chosen in such a way that the resulting assignment π_Y of departures to positions is shunting-free.

As in the last section, we denote by $\mathcal{I}_0$ the set of DTMP instances I for which OPT yields a shunting-free and type-preserving solution. By $\mathcal{I}_2$, we denote the set of DTMP instances for which OPT yields a solution that needs at least two type mismatches. Recall that in this thesis, we assume that for all types the number of trams of this type is identical to the number of departures of the same type. Consequently, if the instance does not admit a solution without type mismatches, then at least two type mismatches are necessary. This implies that $\mathcal{I} = \mathcal{I}_0 \cup \mathcal{I}_2$ is the set of all DTMP instances.

We apply a similar argument as in Theorem 7.3.10 in order to prove the following theorem:

Theorem 7.3.19: There is no deterministic online algorithm ALG satisfying $cost_{\mathrm{ALG}}(I_N) < \frac{N}{2}$ for every instance $I_N \in \mathcal{I}_0$ consisting of N trams.

Proof: We prove the theorem by introducing an instance $I_N \in \mathcal{I}_0$ for which any arbitrary deterministic online algorithm needs at least $\frac{N}{2}$ type mismatches where N denotes the number of trams.

This instance I_N is defined as follows. We consider $N = 4l$ departures of three types $\{\tau_1, \tau_2, \tau_3\}$ where $l \geq 1$. The N trams that have to be assigned to these departures are stored in two stacks of length $2l$.

The first stack consists of $2l$ trams of the following types beginning with the

bottom position:

$$\begin{aligned}&(\tau_2,\tau_1,\tau_3,\tau_1,\tau_2,\tau_1,\ldots,\tau_3,\tau_1,\tau_2,\tau_1) && \text{if } l \text{ is odd}\\&(\tau_3,\tau_1,\tau_2,\tau_1,\tau_3,\tau_1,\ldots,\tau_3,\tau_1,\tau_2,\tau_1) && \text{if } l \text{ is even}\end{aligned}$$

The second stack differs from the first stack in the way that the role of τ_2 and τ_3 are interchanged. Hence, the second stack consists of $2l$ trams of the following types (beginning with the bottom position):

$$\begin{aligned}&(\tau_3,\tau_1,\tau_2,\tau_1,\tau_3,\tau_1,\ldots,\tau_2,\tau_1,\tau_3,\tau_1) && \text{if } l \text{ is odd}\\&(\tau_2,\tau_1,\tau_3,\tau_1,\tau_2,\tau_1,\ldots,\tau_2,\tau_1,\tau_3,\tau_1) && \text{if } l \text{ is even}\end{aligned}$$

The departure sequence $\mathcal{D}$ constructed by the cruel adversary consists of l subsequences of four departures of the following kind $(\tau_1,\tau_2,\tau_1,\tau_3)$ or $(\tau_1,\tau_3,\tau_1,\tau_2)$. The cruel adversary acts in l phases. At the beginning of each phase, one of these subsequences is chosen depending on the actual top elements of the two stacks. Obviously, for such a departure sequence, any optimal offline algorithm OPT yields a solution without any type mismatches (cf. Figure 7.3.11).

At the beginning of each phase, there are two possibilities for the type configuration of the top elements of stack one and stack two:

1. If one stack is empty, then the top element of the other stack is of type τ_1.
2. If the top element of stack one is of type τ, then the top element of stack two is of type τ, and vice versa. The top elements can be of all three types.

We assume that we are at the beginning of phase $i+1$. At that time, $4i$ trams have already been assigned to the first $4i$ departures (cf. Figure 7.3.13).

If $4l > 4i \geq 2l$, then it is possible that all l trams of one stack are assigned to these departures. In this case, $4i-2l$ trams of the other stack are also already assigned. Since $4i-2l>0$ is even, the top element of the non-empty stacks must be of type τ_1.

If both stacks are non-empty, u trams are taken from stack one and v trams are taken from stack two, where $u+v=4i$. If u and v are even, then both top elements are of type τ_1. If u and v are odd, then either $u \equiv 3 \bmod 4$ and $v \equiv 1 \bmod 4$ or vice versa. By construction, the corresponding trams in stack one and stack two are of the same type. If $u \equiv 3 \bmod 4$, then the top elements are of type τ_3. Otherwise, they are of type τ_2. Based on this observation, the departure sequence is constructed in the following way (cf. Figure 7.3.13). In each phase, the first and the third departure are always of type τ_1.

If one stack is empty, the second and fourth departure of a phase are always chosen in such a way that the types do not match with the corresponding types at the actual top positions of the remaining stack. As a consequence, ALG fails on every second request in this phase (and in all following phases).

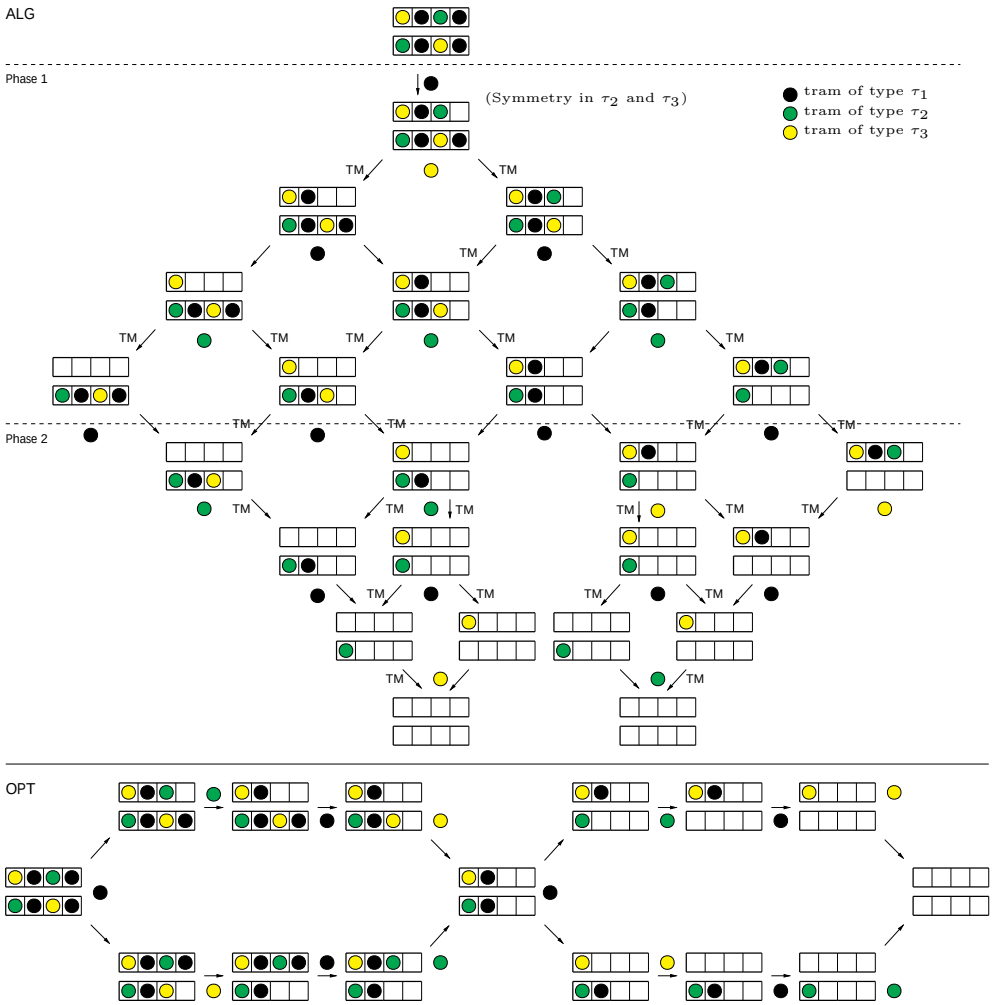

Figure 7.3.13: The instance for $l = 2$ for which ALG needs at least $2l$ type mismatches (denoted by TM) whereas OPT yields a type-preserving solution. Because of symmetry in τ_2 and τ_3, we consider only one of the two possible cases for phase 1.

If the top elements are of type τ_2, then the type sequence of the phase is $(\tau_1, \tau_3, \tau_1, \tau_2)$. Analogously, if the top elements are of type τ_3, then the type sequence is $(\tau_1, \tau_2, \tau_1, \tau_3)$. In both cases, ALG needs at least two type mismatches for this phase since ALG fails on the first departure and on the second departure of this phase.

In the case that the top elements are both of type τ_1, the chosen type sequence depends on the action of ALG. The first departure in the phase is of type τ_1. For this departure, ALG chooses the (matching) tram out of a stack. The type of the resulting top element of this stack is either τ_2 or τ_3. In the first case, we

choose $(\tau_1, \tau_3, \tau_1, \tau_2)$. In the second case, we choose $(\tau_1, \tau_2, \tau_1, \tau_3)$. In both cases, ALG needs at least two type mismatches to serve the departures of this phase since ALG fails on the second and on the fourth departure of this phase.

Consequently, ALG needs at least two type mismatches for serving the departures of each phase. Since there are l phases, ALG constructs an assignment requiring at least $2l = \frac{N}{2}$ type mismatches for instance $I \in \mathcal{I}_0$. ❑

Theorem 7.3.19 gives us a lower bound of $\frac{N}{2}$ on the competitiveness of any arbitrary deterministic online algorithm for DTMP for instances of $\mathcal{I}_0$.

For instances in $\mathcal{I}_2$, we yield the following result:

Theorem 7.3.20: There is no deterministic online algorithm ALG satisfying $cost_{\mathrm{ALG}}(I_N) < \frac{N}{2}$ for every instance $I_N \in \mathcal{I}_2$ where N denotes the number of trams.

Proof: We prove the theorem by giving a class of instances of $\mathcal{I}_2$ for which any arbitrary deterministic online algorithm yields a solution requiring at least $\frac{N}{2}$ type mismatches whereas an optimal offline algorithm only needs two type mismatches.

The instance I_N consists of two stacks of length $\frac{N}{2}$, N even. The stacks are given as follows beginning with the bottom positions:

$$\begin{array}{ll} \text{stack 1} & (\tau_{\frac{N}{2}}, \tau_{\frac{N}{2}-1}, \ldots, \tau_3, \tau_1, \tau_2) \\ \text{stack 2} & (\tau_{N-1}, \tau_{N-2}, \ldots, \tau_{\frac{N}{2}+1}, \tau_1) \end{array}$$

We start generating $\mathcal{D}$ by giving a departure of type τ_1. The arbitrary online algorithm, denoted by ALG, has two possibilities to assign a tram to this departure. First, ALG can decide to assign the tram of type τ_2 on top of stack 1. Secondly, ALG can choose the matching tram of type τ_1 at the position of the stack 2.

In the first case, we continue generating $\mathcal{D}$ with $\frac{N}{2}+1$ departures of the types corresponding to the trams of stack 2 beginning with a tram of type $\tau_{\frac{N}{2}+1}$. ALG has to assign a tram of non-matching type to all these departures. Next, we give departures of types corresponding to the trams of stack 1. We start with a departure of type τ_1. Then, we give a departure of type τ_2. We continue with the departures of types τ_3 to $\tau_{\frac{N}{2}}$. Hence, the type sequence of $\mathcal{D}$ is as follows (beginning with d_1):

$$(\tau_1, \tau_{\frac{N}{2}+1}, \tau_{\frac{N}{2}+2}, \ldots, \tau_{N-1}, \tau_1, \tau_2, \tau_3, \tau_4, \ldots, \tau_{\frac{N}{2}})$$

ALG fails at least on the first $\frac{N}{2}$ departures. OPT starts with first assigning all trams of stack 2 and then assigned the trams of stack 1. Hence, OPT needs only two type mismatches for the departures $d_{\frac{N}{2}+1}$ (of type τ_1) and $d_{\frac{N}{2}+2}$ (of type τ_2).

In the second case where ALG has chosen for the first departure the tram at the top position of stack 2, we choose the types of $\mathcal{D}$ as follows (beginning with d_1):

$$(\tau_1, \tau_1, \tau_3, \tau_4, \ldots, \tau_{\frac{N}{2}}, \tau_2, \tau_{\frac{N}{2}+1}, \tau_{\frac{N}{2}+2}, \ldots, \tau_{N-1})$$

OPT starts with assigning the trams of stack 1 followed by the trams of stack 2 such that OPT needs two type mismatches: one for d_1 (of type τ_1) and one for $d_{\frac{N}{2}+1}$ (of type τ_2). To d_1, ALG has assigned the tram of type τ_1 at the top position of stack 2. For d_2, ALG has two possibilities: a tram of type τ_2 and a tram of type $\tau_{\frac{N}{2}+1}$. In both cases, ALG needs a type mismatch for d_2 as well as for at least the next $\frac{N}{2} - 1$ departures.

We obtain that in all cases we force ALG to need at least $\frac{N}{2}$ type mismatches whereas OPT only requires two type mismatches. ❑

Theorem 7.3.19 and Theorem 7.3.20 imply the following corollary:

Corollary 7.3.21: No deterministic online algorithm for DTMP is (c, d) – competitive where $c < \frac{N}{4}$ or $d < \frac{N}{2}$. (N denotes the number of trams.)

For any algorithm, an upper bound on the number of type mismatches is given by N. This happens if each departure is served by a tram of nonmatching type. Consequently, an upper bound on the competitive ratio c for any arbitrary deterministic online algorithm on instances of $\mathcal{I}_2$ is $\frac{N}{2}$ since OPT fails a least twice on instances of $\mathcal{I}_2$.

Randomization

By the same argumentation as in the previous section, we obtain the following result.

Corollary 7.3.22: No randomized online algorithm for DTMP is (c, d) – competitive against an oblivious adversary where $c < \frac{N}{8}$ or $d < \frac{N}{4}$. (N denotes the number of trams.)

7.3.4 Online Algorithms for DTMP

By Corollary 7.3.21, we observe that no deterministic online algorithm can be better than $\frac{N}{4}$-competitive for instances belonging to $\mathcal{I}_2$ and not better than $\frac{N}{2}$-competitive for instances in $\mathcal{I}_0$. It remains to show whether or not there is an online algorithm with worst-case performance of $\frac{N}{2}$ on $\mathcal{I}_0$ and whether or not the competitive ratio of $\frac{N}{4}$ (or the upper bound $\frac{N}{2}$) on $\mathcal{I}_2$ is tight.

In the following, we restrict ourselves to instances in $\mathcal{I}_0$. Since OPT yields a type-preserving solution for DTMP, there must be a tram having the same type as

the first departure at the top position of one stack. Hence, any online algorithm that matches the first departure with a tram of the same type is $(\frac{N}{2}, N-1)$-competitive.

The greedy algorithm **GREEDY-DTMP** is defined by the following main loop which is iterated for each departure:

GREEDY-DTMP 7.3.23:

> If there is a top-element of suitable type, then assign this tram to the actual departure. Otherwise, choose a tram causing a type mismatch for the actual departure.
> Ties are broken in a fixed deterministic way. We always choose the stack with the smallest index number.

■

Theorem 7.3.24: GREEDY-DTMP is not (c, d) – competitive if $c < \frac{N}{2}$ or $d < N - 2$.

Proof: We start with considering instances in $\mathcal{I}_0$. Beginning with the bottom positions, we introduce a two stack instance of N trams belonging to $\mathcal{I}_0$ ($N > 0$, N even):

$$\begin{array}{ll} \text{stack 1:} & (\tau_{\frac{N}{2}}, \ldots, \tau_4, \tau_3, \tau_2, \tau_1) \\ \text{stack 2:} & (\tau_{\frac{N}{2}+1}, \ldots, \tau_5, \tau_4, \tau_3, \tau_1) \end{array}$$

Both stacks have a length of $\frac{N}{2}$. The departure sequence $\mathcal{D}$ is defined as follows:

$$(\tau_1, \tau_3, \tau_4, \tau_5, \ldots, \tau_{\frac{N}{2}+1}, \tau_1, \tau_2, \tau_3, \tau_4, \ldots, \tau_{\frac{N}{2}})$$

This instance is in $\mathcal{I}_0$. OPT assigns the trams of the second stack first, followed by the trams of the first stack. GREEDY-DTMP starts with stack one. In almost each step except of step one and step $\frac{N}{2}+1$, both stacks have a non-matching tram on-top so that GREEDY-DTMP first assigns the trams of stack one and then the trams of stack two. Consequently, all departures except of the first one and except of departure $d_{\frac{N}{2}+1}$ (having type τ_1) are served by a non-matching tram. Hence, GREEDY-DTMP yields an assignment with $N-2$ type mismatches for this instance.

Next, we introduce an instance of $\mathcal{I}_2$ for which GREEDY-DTMP yields N type mismatches, i.e., to each departure GREEDY-DTMP assigns a tram of non-matching type. The instance consists of two stacks having length $\frac{N}{2}$ and of N

trams, $N \geq 2$, N even. The types of the trams stored in the stacks are given as follows, beginning with the bottom positions:

$$\begin{array}{ll} \text{stack 1} & (\tau_2, \tau_2, \ldots, \tau_2, \tau_1) \\ \text{stack 2} & (\tau_4, \tau_4, \ldots, \tau_4, \tau_3) \end{array}$$

The departure sequence $\mathcal{D}$ is chosen in advance and given by the following type sequence starting with d_1:

$$(\tau_4, \tau_4, \ldots, \tau_4, \tau_1, \tau_2, \tau_2, \ldots, \tau_2, \tau_3)$$

OPT needs two type mismatches for $\mathcal{D}$: one for d_1 and one for d_N. (To d_1, OPT assigns the tram of type τ_3. To d_N, OPT assigns the tram of type τ_4 stored at the bottom position of stack 2.) To each departure, GREEDY-DTMP assigns a tram of non-matching type, since GREEDY-DTMP assigns to d_1 the tram of type τ_1 at the top position of stack 1. To d_2 (and to the following $\frac{N}{2} - 2$ departures), GREEDY-DTMP also assigns the tram (of non-matching) type stored at the actual top position of stack 1. After these $\frac{N}{2}$ departures, stack 1 is empty. GREEDY-DTMP fails also for the remaining $\frac{N}{2}$ departures, since the only tram in stack 2 matching to one of these departures (i.e., the last departure) is stored at the top position of stack 2.

In conclusion, GREEDY-DTMP is forced to require $N - 2$ type mismatches for a class of instances in $\mathcal{I}_0$. Additionally, GREEDY-DTMP needs N type mismatches for a class of instances in $\mathcal{I}_2$ for which OPT requires only two type mismatches. ❑

Next, we consider the following class of (deterministic) online algorithms. By $\mathcal{C}_1$, we define the class of **type matching algorithms**, i.e., the class of deterministic online algorithms for DTMP that choose a tram of suitable type for a departure if possible. If this is impossible, then an algorithm of $\mathcal{C}_1$ assigns a tram of non-matching type in a fixed deterministic way. If there are only trams of non-matching type, an online algorithm of $\mathcal{C}_1$ is restricted to choose the tram stored at the top position of the (non-empty) stack having the lowest index. If there are some trams of suitable, an online algorithm chooses one of these trams but is not restricted to choose the tram from a particular stack. GREEDY-DTMP belongs to $\mathcal{C}_1$ because GREEDY-DTMP always chooses a tram of suitable if possible. Moreover,

Corollary 7.3.25: No online algorithm of $\mathcal{C}_1$ is (c, d)-competitive where $c < \frac{N}{2}$ and $d < N - 2$.

Proof: For the worst case examples of Theorem 7.3.24, every online algorithm ALG of $\mathcal{C}_1$ performs similar to GREEDY-DTMP.

First, we consider the class of instances in $\mathcal{I}_0$. The stacks are given by

$$\begin{aligned} &\text{stack 1:} && (\tau_{\frac{N}{2}}, \dots, \tau_4, \tau_3, \tau_2, \tau_1) \\ &\text{stack 2:} && (\tau_{\frac{N}{2}+1}, \dots, \tau_5, \tau_4, \tau_3, \tau_1) \end{aligned}$$

ALG chooses either the top element of stack 1 or the top element of stack 2. In the first case, $\mathcal{D}$ is defined as follows (beginning with d_1):

$$(\tau_1, \tau_3, \tau_4, \dots, \tau_{\frac{N}{2}+1}, \tau_1, \tau_2, \tau_3, \dots, \tau_{\frac{N}{2}})$$

ALG begins with assignment the trams of stack 1. Then, ALG assigns the trams of stack 2. ALG needs $N-2$ type mismatches whereas OPT yields a type-preserving assignment.

In the second case where ALG chooses the top element of stack 2, we let $\mathcal{D}$ start with the types of stack 1, i.e.,

$$(\tau_1, \tau_2, \tau_3, \dots, \tau_{\frac{N}{2}}, \tau_1, \tau_3, \tau_4, \dots, \tau_{\frac{N}{2}+1})$$

ALG starts with the trams of type τ_1 at the top position of stack 2. ALG continues with the trams of stack 1 followed by remaining trams in stack 2. ALG needs $N-2$ type mismatches whereas OPT again yields a type-preserving assignment.

If the instance belongs to $\mathcal{I}_2$, then we give the same instance as in the proof of Theorem 7.3.24. ALG behaves as GREEDY-DTMP and fails for every departure. ❑

Chapter 8

Real-Time Tram Dispatch

In the last two chapters, we focused on online dispatching problems. In this chapter, we discuss the variations of the online tram dispatch problems which arise in real-world depots. In contrast to the online dispatch problem, in the real-time situation the computation time is crucial. Usually, there are only a few minutes between two consecutive arrivals of trams at the depot. Hence, the decision to which depot location the actual arriving tram should be assigned has to be made within this time bound. In the following, we introduce an approach that uses the techniques and models of the previous sections and meets the requirements of the practical dispatch process.

8.1 Arrival of Trams in Real-Time

After a tram has served its round trip, it starts its pull-in trip on the way back to the depot. The pull-in time depends on the length of pull-in trip and on external influences, for instance the traffic situation. The pull-in trip leads the tram back to the depot on the shortest path from the station served as last. Since the shortest paths of several trams often coincide, the arrival order at the depot depends significantly on delays in the daily schedule.

After finishing the round trip, the tram driver transmits a message that he is on the way back to the depot. During the pull-in time of the tram, a suitable location and a possible round trip for the next schedule period has to be found. The time bound for the corresponding decision lies between two and five minutes. Moreover, the time available may be influenced by subsequent messages from other trams that give more information about the concrete arrival order of trams.

A change in the arrival order of trams may have influences on the number of shunting movements needed to store the trams as originally planned (and to let them leave the depot at departure). Consequently, the assignments of trams to positions and of trams to departures must be updated in such a way that the new solution “minimizes” the number of shunting movements (or type mismatches).

8.2 Real-Time Algorithms

Based on the scheduled arrival and departure of trams, we generate an assignment of trams to positions and to departures (using the methods of Chapter 4). As long as the trams arrive as scheduled, the trams are assigned in accordance with this predetermined solution. A solution update becomes necessary if the arrival order changes in the way that a tram of different type than expected arrives at the depot. In the case that this tram can be assigned as planned to the location in the depot without shunting, the update of the arrival sequence does not require an update of the solution. Otherwise, we have three possibilities to react on such a real-time effect:

1. By a local decision without a complete (or partial) revision of the solution structure, we can decide to store the tram at another location.

2. We can perform a partial update of the assignments by recomputing the assignments for the positions involved in the change of the arrival sequence.

3. We can decide to do a complete revision of the assignment of (forthcoming) trams to positions and departures.

In the last case, we have to solve a problem equivalent to (TDP) (or (TMP) respectively) that have been introduced in Chapter 4. Since these problems are $\mathcal{NP}$-hard, a complete revision is impossible in the real-time dispatch process.

The LIFO heuristic introduced in Chapter 4.5 is a real-time algorithm that proceeds with local decisions. For each arriving tram, LIFO determines among all unassigned departures the last unassigned departure of the same type and assigns the tram to this departure. Then, LIFO starts with searching for a suitable position to which the tram should be assigned. While determining the assignment for the actual tram, LIFO does not regard further arrivals. The performance of LIFO has been examined in Section 4.5. In the following, we will restrict ourselves to real-time algorithms that compute a partial update of the solution.

Based on the scheduled arrival sequence represented by $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, the set of positions $\mathcal{P}$, and the sequence of departures $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, we assume that we are given an (optimal) assignment of trams to positions $\pi_X : \mathcal{A} \to \mathcal{P}$ and an (optimal) assignment of departures to positions $\pi_Y : \mathcal{D} \to \mathcal{P}$. π_X and π_Y are assumed to be type-preserving.

Let i be the (smallest) index of the arrival sequence where the actual arriving tram differs from a_i. Moreover, we assume that we know an interval $[i, i + \delta]$ in which the actual arrival sequence differs from $\mathcal{A}$. We assume that the trams $a_j \in \mathcal{A}$ with $j < i$ have already arrived at the depot and that there are no further changes known for a_k with $k > i + \delta$. The resulting arrival sequence is denoted by $\mathcal{A}'$ and the partial arrival sequence corresponding to $[i, i + \delta]$ is denoted by $\mathcal{A}'_\delta$.

Since a revision of the assignment of a_1 to a_{i-1} would lead to a rearrangement of these trams and may lead to shunting movements, we require that π_X is left unchanged for all trams in $\{a_1, \ldots, a_{i-1}\}$. Additionally, we force that π_X is also left unchanged for all trams in $\{a_{i+\delta+1}, \ldots, a_N\}$, because we do not want to revise π_X completely due to the lack of time.

Analogously to the notation for π_X, we denote $\mathcal{P}_\delta$ as the set of positions to which $a_i, a_{i+1}, \ldots, a_{i+\delta}$ are supposed to be assigned, i.e., $\mathcal{P}_\delta = \{p \in \mathcal{P} \mid p = \pi_X(a_j), a_j \in \{a_i, \ldots, a_{i+\delta}\}\}$. By $\mathcal{D}_\delta$, we denote the set of departures that are assigned to $\mathcal{P}_\delta$ (according to π_Y), i.e., $\mathcal{D}_\delta = \{d \in \mathcal{D} \mid \pi_Y(d) \in \mathcal{P}_\delta\}$.

The corresponding (restricted) assignments of π_X and π_Y that have to be updated are denoted by $\pi_X^\delta : \mathcal{A}'_\delta \to \mathcal{P}_\delta$ and $\pi_Y^\delta : \mathcal{D}_\delta \to \mathcal{P}_\delta$.

Hence, two possible update calculations are:

1. Recompute π_X^δ and π_Y^δ by solving the corresponding (TDP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}_\delta$. (**Real-time algorithm (TDP-n)**)

2. Recompute π_X^δ and π_Y^δ by solving the corresponding (TMP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}_\delta$. (**Real-time algorithm (TDP-n)**)

In both cases, we are given π_X and π_Y for all other trams in $\mathcal{A}\backslash\mathcal{A}'_\delta$ and all other departures in $\mathcal{D}\backslash\mathcal{D}_\delta$ which have to be taken into consideration while updating π_X^δ and π_Y^δ. By the extension '**n**', we denote that we do not compute a complete revision of π_Y.

A more global update is the recomputation of the complete assignment of trams to departures. This results in:

1. Solving the corresponding (TDP) restricted to π_X^δ and π_Y where π_X is given for all other trams in $\mathcal{A}\backslash\mathcal{A}'_\delta$. (**Real-time algorithm (TDP-y)**)

2. Solving the corresponding (TMP) restricted to π_X^δ and π_Y where π_X is given for all other trams in $\mathcal{A}\backslash\mathcal{A}'_\delta$. (**Real-time algorithm (TMP-y)**)

By the extension '**y**', we denote explicitly that we compute a complete revision of π_Y. Since these problems are generalizations of DTDP and DTMP respectively, the resulting problems are $\mathcal{NP}$-hard.

Another alternative is to update π_Y only for those positions located in stacks $\mathcal{P}_r$ that will be involved in an update of π_X^δ, $1 \leq r \leq R$. The corresponding set of positions, denoted by $\mathcal{P}'_\delta$, is defined as follows: $\mathcal{P}'_\delta = \bigcup_r \{p \in \mathcal{P}_r \mid \mathcal{P}_r \cap \mathcal{P}_\delta \neq \emptyset\}$. The set of departures assigned to $\mathcal{P}'_\delta$ is denoted by $\mathcal{D}'_\delta$ and defined as $\mathcal{D}'_\delta = \{d \in \mathcal{D} \mid \pi_Y(d) \in \mathcal{P}'_\delta\}$. By $\pi_Y^{\delta'} : \mathcal{D}'_\delta \to \mathcal{P}'_\delta$, we denote the corresponding assignment of departures to positions in $\mathcal{P}'_\delta$.

Once again, we have two possibilities to determine updated assignments π'_X and π'_Y:

1. First, we can solve the (TDP) restricted to π_X^δ and $\pi_Y^{\delta'}$ where π_X is given for the trams in $\mathcal{A}\backslash\mathcal{A}'_\delta$ and π_Y is given for the departures in $\mathcal{D}\backslash\mathcal{D}'_\delta$. (**Real-time algorithm (TDP-s)**)

2. Secondly, we can solve the corresponding (TMP) for the same instance. (**Real-time algorithm (TMP-s)**)

By the extension '**s**', we denote that we compute an updated assignment of departures to the positions in the stacks which are involved in an update of π_X^δ.

Summarizing, we consider six possibilities to compute a partial update of π_X and π_Y. With each possibility, we identify a real-time algorithm that solves the corresponding problem in its main loop in the case that the actual arriving tram differs from the tram which has been expected to arrive as the next.

- Solving the corresponding (TDP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}_\delta$ resulting in an update of π_X^δ and π_Y^δ: **Real-time algorithm (TDP-n)**
- Solving the corresponding (TMP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}_\delta$ resulting in an update of π_X^δ and π_Y^δ: **Real-time algorithm (TMP-n)**
- Solving the corresponding (TDP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}$ resulting in an update of π_X^δ and π_Y: **Real-time algorithm (TDP-y)**
- Solving the corresponding (TMP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}$ resulting in an update of π_X^δ and π_Y: **Real-time algorithm (TMP-y)**
- Solving the corresponding (TDP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}'_\delta$ resulting in an update of π_X^δ and $\pi_Y^{\delta'}$: **Real-time algorithm (TDP-s)**
- Solving the corresponding (TMP) restricted to $\mathcal{A}'_\delta$ and $\mathcal{D}'_\delta$ resulting in an update of π_X^δ and $\pi_Y^{\delta'}$: **Real-time algorithm (TMP-s)**

8.3 The Real-Time Scenarios

We consider two real-time scenarios. In the first scenario, we assume that the dispatcher only gets information about the next arriving tram. This tram is chosen randomly among the next Δ trams. We assume that the actual arriving tram has index i and that the chosen tram has index $i+\delta$ in the sequence of arrivals. Then, the arrival sequence is changed in the following way: $a_i, a_{i+1}, \ldots, a_{i+\delta}$ is changed to $a_{i+\delta}, a_i, a_{i+1}, \ldots, a_{i+\delta-1}$. The remaining arrival sequence is left unchanged. This real-time scenario is similar to the online setting of the previous chapter. Only the information about the next arrival is given to the real-time (online) algorithm. The frequency of such changes in the arrival sequence is chosen randomly in the following way. For each arrival, we observe the value of a

uniformly distributed $[0, 1]$ random variable. If that value is larger than a given threshold parameter, then the arrival sequence is changed as described above.

In the second real-time scenario, we consider the case that the arrival sequence is changed completely in the interval $[i, i + \delta]$. In this partial sequence the order of arrivals is completely inverted. The partial sequence $a_i, a_{i+1}, \ldots, a_{i+\delta-1}, a_{i+\delta}$ is changed to $a_{i+\delta}, a_{i+\delta-1}, \ldots, a_{i+1}, a_i$. How often such changes are generated for an instance is chosen arbitrarily.

In both scenarios, we proceed as follows. Based on an initial solution, we compute the updated assignment for the new arrival sequence. Next, this new solution is used in the possible further iterations and update steps. Note that the updated arrival sequences may overlap in some iterations.

8.4 Computational Results

We apply the six real-time algorithms described in Section 8.2 to the practical instances of Braunschweig (bs) and Karlsruhe (ka) and to one random instance of size 40_8_6, i.e, 40 trams, 8 stacks (of length 5), and 6 types.

For scenario 1, TDP-n, TMP-n, and TDP-s yield solutions which require only a few shunting movements. TDP-n needs significantly less computation time than TMP-n. For $\delta \geq 8$, TDP-s is faster than TMP-n. In the most cases, TDP-n and TDP-s also yield better solutions than TDM-n (cf. Table 8.1 – Table 8.4).

For the Braunschweig instances, it is also possible to apply TDP-y which results in a complete update of the assignment of departures. For the Karlsruhe instances and for large δ, this becomes impossible within the given time limit of five minutes per iteration. Additionally, TMP-s and TMP-y often fail in computing solutions within the given time limit. In particular, this holds for the Karlsruhe instances where more trams have to be dispatched.

For scenario 2, we achieve similar results. In this scenario, we restrict ourselves to the real-time algorithms TMP-n, TMP-y, TDP-n, and TDP-y. Table 8.5 shows the results for TMP-n and TMP-y. The results for TDP-n and TDP-y are presented in Table 8.6. We observe that we can solve the real-time instances by applying TMP-n and TDP-n. Once again, TDP-n outperforms TMP-n, in particular for large δ. TDP-y yields acceptable results for the smaller instances and for small values of δ. In general, TMP-y fails in computing solution within the given time bound of 300 seconds per iteration.

The solutions obtained by the real-time algorithms are compared with the solutions that the LIFO heuristic (cf. Definition 4.5.2) yields for the final instances, i.e., the instances which result after all trams have arrived. The solution value obtained by LIFO can be regarded as an upper bound on the optimal number of shunting movements (or by applying Theorem 4.3.13 as an upper bound on the optimal number of type mismatches). LIFO itself is a fast online algorithm which assigns the trams step by step without regarding the remaining sequence of

arrivals. Hence, LIFO may also be applied to the real-time scenarios. Since LIFO does not regard the predetermined assignment of trams to positions, LIFO may change completely the predetermined solution which may result in more shunting movements. For the considered instances, our computational results show that LIFO yields good results.

instance	Δ	δ	#	TDP-n		TMP-n		TDP-s	
				SM	CPU [s]	TM	CPU [s]	SM	CPU [s]
bs.mo-do	3	[1,3]	8	1	0.15	4	0.40	0	0.69
	5	[1,5]	8	0	0.21	0	3.09	0	1.27
	8	[1,8]	8	0	0.89	0	331.34	0	5.61
	10	[1,10]	8	0	3.53	0	603.14	0	29.18
	12	[1,12]	8	0	9.09	0	604.15	0	141.96
	15	[1,14]	8	0	24.03	0	607.47	0	100.08
bs.fr	3	[1,3]	8	0	0.11	0	0.36	0	0.53
	5	[1,5]	8	0	0.34	2	2.27	0	0.85
	8	[1,8]	8	0	0.83	2	309.77	2	8.98
	10	[1,10]	8	2	1.07	2	402.02	2	28.67
	12	[1,12]	8	1	4.43	0	602.75	0	56.91
	15	[1,14]	8	0	21.09	0	613.95	0	53.99
bs.sa	3	[1,3]	8	0	0.15	0	0.35	0	0.44
	5	[1,5]	8	1	0.18	0	5.90	0	1.84
	8	[1,8]	8	0	0.43	2	321.16	0	8.85
	10	[1,10]	8	0	2.49	0	545.12	0	35.41
	12	[1,12]	8	0	1.87	0	604.51	0	50.66
	15	[1,14]	8	0	1.84	2	385.15	0	129.02
bs.so	3	[1,3]	1	0	0.04	0	0.07	0	0.07
	5	[1,5]	3	0	0.03	0	0.08	0	0.07
	8	[1,2]	2	0	0.05	0	0.09	0	0.08
	10	[1,3]	3	0	0.05	0	0.08	0	0.09
	12	[1,12]	3	0	0.07	2	0.12	0	0.15
	15	[1,4]	3	0	0.05	2	0.11	0	0.14

Table 8.1: Real-time scenario 1: Comparison of the performance of the real-time algorithms TDP-n, TMP-n, and TDP-s. Results for real-world instances. Final objective values after several update steps. Cumulative computation times for # iterations. (SM denotes the number of shunting movements, TM denotes the number of type mismatches.)

instance	Δ	δ	#	TMP-s		TDP-y		TMP-y	
				TM	CPU [s]	SM	CPU [s]	TM	CPU [s]
bs.mo-do	3	[1,3]	8	2	4.69	0	124.97	0	2042.67
	5	[1,5]	8	0	30.01	0	96.96	5	1664.48
	8	[1,8]	8	0	913.65	0	135.45	-	>2400.00
	10	[1,10]	8	0	919.47	0	279.63	-	>2400.00
	12	[1,12]	8	-	>2400.00	0	257.88	-	>2400.00
	15	[1,14]	8	-	>2400.00	0	514.79	-	>2400.00
bs.fr	3	[1,3]	8	0	3.92	0	79.24	2	1069.01
	5	[1,5]	8	0	30.16	0	70.69	2	1577.11
	8	[1,8]	8	2	732.29	0	255.59	-	>2400.00
	10	[1,10]	8	4	957.11	0	545.10	-	>2400.00
	12	[1,12]	8	-	>2400.00	0	185.57	-	>2400.00
	15	[1,14]	8	-	>2400.00	0	211.55	-	>2400.00
bs.sa	3	[1,3]	8	0	4.78	0	45.85	5	1332.40
	5	[1,5]	8	0	32.47	0	67.93	4	1403.73
	8	[1,8]	8	-	>2400.00	0	78.04	-	>2400.00
	10	[1,10]	8	-	>2400.00	0	111.93	-	>2400.00
	12	[1,12]	8	-	>2400.00	0	134.23		>2400.00
	15	[1,14]	8	-	>2400.00	0	192.94	-	>2400.00
bs.so	3	[1,3]	2	0	0.19	0	0.59	3	4.56
	5	[1,5]	3	0	0.19	0	0.59	4	4.60
	8	[1,2]	2	0	0.32	0	0.73	3	6.00
	10	[1,3]	3	0	0.39	0	0.86	3	4.33
	12	[1,12]	3	2	0.85	0	0.90	2	19.58
	15	[1,4]	3	2	0.83	0	0.97	2	19.61

Table 8.2: Real-time scenario 1: Comparison of the performance of the real-time algorithms TMP-s, TDP-y, and TMP-y. Results for real-world instances. Final objective values after several update steps. Cumulative computation times for # iterations. (SM denotes the number of shunting movements, TM denotes the number of type mismatches.)

instance	Δ	δ	#	TDP-n		TMP-n		TDP-s	
				SM	CPU [s]	TM	CPU [s]	SM	CPU [s]
ka.26	3	[1,3]	10	2	0.17	2	1.26	0	0.58
	5	[1,5]	10	0	0.33	0	8.19	0	1.81
	8	[1,8]	10	0	0.43	4	707.14	0	16.93
	10	[1,10]	10	0	3.20	2	907.31	0	86.31
	12	[1,12]	10	0	3.33	2	938.43	0	160.87
	15	[1,15]	10	0	21.33	8	1226.53	0	490.26
ka.27	3	[2,3]	4	0	0.06	0	0.21	0	1.25
	5	[3,5]	4	0	0.10	0	0.31	0	2.51
	8	[3,8]	4	0	0.17	0	36.74	0	5.49
	10	[3,9]	4	0	0.22	0	300.73	0	24.58
	12	[3,9]	4	0	0.55	0	325.86	0	38.67
	15	[3,9]	4	0	0.80	0	600.62	0	45.89
ka.28	3	[1,3]	10	3	0.20	6	0.87	2	0.54
	5	[1,5]	10	0	0.47	0	6.48	1	0.99
	8	[1,8]	10	0	0.93	0	809.53	0	16.98
	10	[1,10]	10	1	8.87	0	902.82	0	23.65
	12	[1,12]	10	0	14.92	0	915.01	0	91.89
	15	[1,15]	10	1	167.80	7	935.63	1	184.40
ka.29	3	[1,3]	10	1	0.24	2	1.12	0	0.58
	5	[1,5]	10	0	0.25	4	8.32	0	1.18
	8	[1,8]	10	1	2.84	2	413.50	0	11.32
	10	[1,10]	10	0	3.86	0	908.22	0	26.27
	12	[1,12]	10	0	17.16	2	907.59	0	92.26
	15	[1,15]	10	0	68.64	6	924.44	0	185.59

Table 8.3: Real-time scenario 1: Comparison of the performance of the real-time algorithms TDP-n, TMP-n, and TDP-s. Results for real-world instances. Final objective values after several update steps. Cumulative computation times for # iterations. (SM denotes the number of shunting movements, TM denotes the number of type mismatches.)

instance	Δ	δ	#	TMP-s		TDP-y		TMP-y	
				SM	CPU [s]	TM	CPU [s]	SM	CPU [s]
ka.26	3	[1,3]	10	0	4.94	4	>3000.00	-	>3000.00
	5	[1,5]	10	0	88.46	5	>3000.00	-	>3000.00
	8	[1,8]	10	2	927.34	-	>3000.00	-	>3000.00
	10	[1,10]	10	-	>3000.00	-	>3000.00	-	>3000.00
	12	[1,12]	10	-	>3000.00	-	>3000.00	-	>3000.00
	15	[1,15]	10	-	>3000.00	-	>3000.00	-	>3000.00
ka.27	3	[1,3]	4	0	11.31	0	50.48	0	377.57
	5	[1,5]	4	0	137.50	0	46.43	0	579.98
	8	[1,8]	4	2	334.81	0	54.72	-	>1200.00
	10	[1,10]	4	-	>1200.00	0	54.49	-	>1200.00
	12	[1,12]	4	-	>1200.00	0	142.05	-	>1200.00
	15	[1,15]	4	-	>1200.00	0	131.87	-	>1200.00
ka.28	3	[1,3]	10	5	2.65	1	1560.35	-	>3000.00
	5	[1,5]	10	4	58.97	0	1825.88	-	>3000.00
	8	[1,8]	10	2	907.68	0	2439.59	-	>3000.00
	10	[1,10]	10	-	>3000.0	1	2770.10	-	>3000.00
	12	[1,12]	10	-	>3000.0	-	>3000.00	-	>3000.0
	15	[1,15]	10	-	>3000.0	-	>3000.00	-	>3000.0
ka.29	3	[1,3]	10	6	12.22	0	2707.63	-	>3000.00
	5	[1,5]	10	4	155.42	5	>3000.00	-	>3000.00
	8	[1,8]	10	4	914.88	-	>3000.00	-	>3000.00
	10	[1,10]	10	-	>3000.00	-	>3000.00	-	>3000.00
	12	[1,12]	10	-	>3000.00	-	>3000.00	-	>3000.00
	15	[1,15]	10	-	>3000.00	-	>3000.00	-	>3000.00

Table 8.4: Real-time scenario 1: Comparison of the performance of the real-time algorithms TMP-s, TDP-y, and TMP-y. Results for real-world instances. Final objective values after several update steps. Cumulative computation times for # iterations. (SM denotes the number of shunting movements, TM denotes the number of type mismatches.)

instance	δ	TMP-n			TMP-y		
		TM	CPU [s]	LIFO	TM	CPU [s]	LIFO
bs.mo-do	5	0	0.94	0	-	> 300.00	0
	10	0	300.22	0	-	> 300.00	0
	15	-	> 300.00	0	-	> 300.00	0
bs.fr	5	0	0.63	0	-	> 300.00	0
	10	0	300.21	0	-	> 300.00	0
	15	-	> 300.00	0	-	> 300.00	0
ka.26	5	0	1.13	0	-	> 300.00	0
	10	5	300.88	0	-	> 300.00	0
	15	-	> 300.00	0	-	> 300.00	0
ka.27	5	0	0.99	0	0	301.94	0
	10	2	225.26	0	-	> 300.00	0
	15	6	305.60	0	-	> 300.00	0
40_8_6	5	6	0.27	(1)	-	> 300.00	(1)
	10	0	127.57	0	-	> 300.00	0
	15	5	303.53	0	-	> 300.00	0

Table 8.5: Real-time scenario 2: Comparison of the performance of the real-time algorithms TMP-n and TMP-y. Results for real-world instances. Final objective values after several update steps. Average computation times. TM denotes the number of type mismatches. LIFO denotes the result obtained by applying LIFO as an offline algorithm to the final instance.

instance	δ	TDP-n			TDP-y		
		SM	CPU [s]	LIFO	SM	CPU [s]	LIFO
bs.mo-do	5	0	0.04	0	0	17.40	0
	10	0	0.76	0	0	42.34	0
	15	0	37.37	0	0	186.38	0
bs.fr	5	0	0.03	0	0	69.77	0
	10	1	1.35	0	0	61.58	0
	15	0	38.90	0	0	153.96	0
ka.26	5	0	0.01	0	0	287.18	0
	10	0	1.18	0	-	> 300.00	0
	15	0	24.59	0	-	> 300.00	0
ka.27	5	0	0.05	0	0	12.20	0
	10	0	3.02	0	1	50.39	0
	15	0	56.27	0	0	125.99	0
40_8_6	5	4	0.06	(1)	7	272.01	(1)
	10	0	1.85	0	-	> 300.00	0
	15	0	157.58	0	-	> 300.00	0

Table 8.6: Real-time scenario 2: Comparison of the performance of the real-time algorithms TDP-n and TDP-y. Results for real-world instances. Final objective values after several update steps. Average computation times. SM denotes the number of shunting movements. LIFO denotes the result obtained by applying LIFO as an offline algorithm to the final instance.

8.5 Conclusion

In this chapter, we introduced six real-time algorithms which are based on the models and results for TDP and TMP. These algorithms differ in the objective function which has to be minimized, i.e., the number of shunting movements or the number of type mismatches. Moreover, the algorithms differ in the computation of the assignment of departures.

All six algorithms are based on a predetermined solution which gives us the assignment of trams to positions and the assignment of departures to the trams stored at the corresponding positions. Such an initial solution can be computed using the exact or heuristic methods introduced in Chapter 4. For instance, an initial solution may be computed by the LIFO heuristic.

The real-time algorithms update the given assignment of trams to departures only for the arriving trams whose arrival order is changed because of some real-time effects. For the remaining trams the assignment to positions is left unchanged.

In accordance with the classification of real-time algorithms of [SPG+97], the real-time algorithms may be regarded as incremental planning algorithms. In the case that we decide to compute a completely new assignment of departures, the corresponding algorithms can be also considered as deliberative planning algorithms.

For the real-world and random instances considered in this chapter, we observe that TDP-n, TDP-s, TMP-n, and TDP-y yield good results for different lengths of changes in the arrival sequence. These four algorithms can be ranked in the following way: TDP-n, TDP-s, TMP-n, and TDP-y. TDP-y should only be applied to small instance or if the update involves only few arriving trams.

The computational results for these algorithms show that we can compute an updated assignment within a time bound of two to five minutes. In the case that a solution is required within less than two minutes, we have to restrict ourselves to real-time algorithm which have a computational complexity independent of the number of trams (or departures), i.e., to TDP-n and TMP-n. In some cases, also TDP-s may be suitable. A constant (or linear) computational complexity seems to be important for such fast solution updates.

The solutions obtained by our algorithms require only a few shunting movements or type mismatches or do not even require any shunting movements or type mismatches. Consequently, these algorithms can find an application in decision support system for tram dispatch.

An alternative is the application of the LIFO heuristic (cf. Definition 4.5.2). LIFO is a polynomial-time online algorithm which yields solutions even for large instances within less than one second of computation time. A disadvantage of LIFO is that LIFO may change the predetermined solution completely. Hence, the final assignment of trams to positions and departures may differ substantially from the initial assignment which was based on the schedule.

In conclusion, we recommend the use of TDP-n and TDP-s. If shunting has to be avoided, TMP-n should be applied. In the case that we may change the initial solution completely, we also recommend the application of the LIFO heuristic.

Chapter 9

Related Problems

In this chapter, we discuss several stacking problems which are in some way related to the tram dispatching problem (TDP). Stacks play an important role in the analysis of logistic systems and in computer science.

In the 1960s, stack sorting problems have been studied by Knuth [Knu68]. Based on this work, several extensions and related stack sorting have been examined, for instance the question in which cases a given permutation is sortable by stacks. This problem has been considered by Even and Itai [EI71], Knuth [Knu68], Rotem [Rot81], West [Wes93], and Zeilberger [Zei92]. These permutation sorting problems are strongly connected with graph coloring problems. In this context, Johnson et al. [GJMP80] and Unger [Ung88, Ung92] examine coloring problems of circular arc graphs.

9.1 Sorting Permutations by Stacks

Knuth [Knu68] considers stacks, queues, and dequeues. He represents a stack as a railway switching network of the following form: Given a tram of numbered railway cars, the cars have to be moved through the switching network so that they leave the switching network in a non-decreasing order according to their numbers. This visualization of a stack has been suggested by Dijkstra (cf. Figure 9.1.1).

Knuth examined in which cases it is possible to sort a given permutation using a stack, i.e., to achieve the identity permutation as output. The stack can be accessed in the following way. We have two possible moves: Either we decide to move a railway car into the stack or we have to remove a railway car from the stack. Knuth showed that a given permutation can be sorted if and only if it contains no **wedge**. A wedge is defined as a subsequence of items that can be numbered by 2,3, and 1 without changing their order. It is obvious that the permutation 2,3,1 cannot be sorted with such a stack (see Figure 9.1.1).

The problem of achieving a certain permutation from a given one through a network of parallel stacks (and queues, respectively) is considered by Even

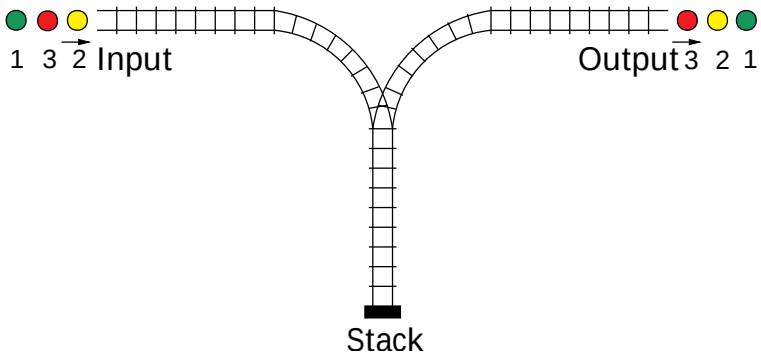

Figure 9.1.1: A stack represented as a railway switching network.

and Itai. [EI71]. Even and Itai focus on two versions of this problem. First, they examine the problem in which the stacks have to filled up until the first item is removed. In this case, the number of stacks which are necessary to sort the given permutation is equal to the chromatic number of the induced permutation graph. This chromatic number can be computed in polynomial-time (see Golumbic [Gol80]). Secondly, they allow to remove items from the stacks before the last item is loaded. This problem turns out to be equivalent to the problem of determining the chromatic number of a union (or overlap) graph. Johnson [GJMP80] showed that this problem is $\mathcal{NP}$-hard in the general case. It remains hard even if we restrict ourselves to a fixed number of at least four stacks [Ung92]. It is decidable in polynomial-time whether or not a given permutation can be sorted using two or three stacks [Ung88]. We will come back to this problem in Section 9.4.

The problem of sorting permutations using a stack is also examined by West [Wes93] and Rotem [Rot81]. West views the stack sorting problem as a mapping from permutations to permutations. He considers those permutations of length n which become sorted after k applications of this function. All permutations are sorted after $n-1$ iterations where $(n-2)!$ permutations require exactly this number of iterations. The number of permutations that can be sorted by one application is equal to the Catalan number $c_n = \frac{1}{n+1}\binom{2n}{n}$. West also characterizes the permutations that can be sorted with two applications of the corresponding function. He shows that a permutation is two-stack sortable if and only if it contains no subsequence of the following types: 2,3,4,1 and 3,2,4,1 in the case that it is not part of 3,5,2,4,1. His conjecture on the number of such two-stack sortable permutations has been proven by Zeilberger [Zei92].

9.2 Sorting Pancakes

A particular stack sorting problem is the so-called **pancake problem**. In this problem, we are given a stack of n pancakes of different size. The pancakes have to be sorted according to the pancake size with the largest pancake at the bottom of the stack. The allowed sorting operation is a **spatula flip** where the spatula is inserted between any two stack positions. The pancakes above the spatula are lifted and replaced in reverse order. The problem is to determine the minimum number of flips needed to sort the pancakes. Gates and Papadimitriou [GP79] identify this problem as a list sorting problem in which the objective is to minimize the number of prefix reversals. They give a lower bound of $\frac{17n}{16}$ and an upper bound of $\frac{5n+3}{3}$ for the worst case.

A variation of this problem, the burnt pancake problem, is considered by Cohen and Blum [CB95]. Cohen and Blum examine the problem where the pancakes have two different sides. Exactly one side of the pancake is burnt. The goal is to sort the pancakes with a minimum number of reversals so that all burnt sides are face-down. Cohen and Blum derive worst-case bounds for the number of reversals. The lower bound is $\frac{3n}{2}$ and the upper bound of $2n - 2$ holds for $n \geq 10$. They conjecture an upper bound of $\frac{23n}{14} + c$ where c is a small constant. This conjecture holds for $n \leq 10$. It is motivated by an assumed worst-case configuration: the inverted identity permutation.

9.3 The Train Marshalling Yard Problem

Dahlhaus et al. [DHMR] consider a railway car rearrangement problem. In this problem, a train consisting of n railway cars arrives at a station. The cars are arranged in a given order and each car has a specified destination.

After the arrival at the station, the cars having the same destination have to be separated from the other. The resulting rearrangement process is carried out at a shunting yard consisting of k sidings. Beginning with the first railway car a_1, each railway car a_i is moved into one of the k sidings such that the train is split into k subtrains. After the last car a_n has been assigned, the new rearranged train is formed by coupling the k subtrains.

Dahlhaus et al. examine the minimum number k^* of sidings needed to rearrange the train. Obviously, k^* is bounded from above by the number of car destinations. Dahlhaus et al. formalize this problem in the following way. We denote by T the number of destinations. Given a partition of $S = \{a_1, a_2, \ldots, a_n\}$ into T disjoint sets $S_1, \ldots, S_T$, we are looking for a permutation $\pi : \{1, \ldots, T\} \rightarrow \{1, \ldots, T\}$. This permutation π has to satisfy the property that the sequence of numbers $1, 2, \ldots, n, \ldots, 1, 2, \ldots, n$ contains all the elements of $S_{\pi(i)}$, then the elements of $S_{\pi(j)}$, $i < j$. Note that, in this sequence, $1, 2, \ldots, n$ is repeated k times.

Dahlhaus et al. show that this problem is $\mathcal{NP}$-hard by giving a reduction from Numerical Matching with Targets Sums (see [GJ79]) to the decision problem whether a solution for a fixed number k of sidings exists. They derive an upper bound of $\lceil \frac{n}{4} + \frac{1}{2} \rceil$ for k^*.

9.4 The Container Stowage Problem (CSP)

Another stack sorting problem occurring in container logistics is considered by Avriel et al. [AP93, APS96, APSW98]. In this problem, a container ship (vessel) is calling several ports. In each port, containers are unloaded and loaded. The containers are stored in stacks on-top of each other. Every time a specified container is unloaded from a stack in a port, all containers standing on-top of this container have to be unloaded, too. Additionally, if these containers have a different destination port they have to reloaded. Since such container shifts are expensive and require time, the objective is to minimize their number.

Avriel and Penn [AP93] present the following binary integer program for a ship with a rectangular bay. By N, we denote the number of ports, by R the number of rows (layers), by C the number of columns (stacks), and by $P = R \cdot C$ the total number of bay positions. We identify the assignment of a container to a bay position (r, c) by the binary variable $x_{ijv}(r, c)$. Here, $x_{ijv}(r, c)$ equals 1 if and only if a container having destination port j is loaded in port i at the bay position (r, c) and is unloaded in port v, $v \leq j$. If $x_{ijv}(r, c) = 1$ for some $j < v$ and some position (r, c), then the corresponding container is unloaded before it reaches the container's destination port. Hence, it has to be reloaded in port v.

The set of ports $\mathcal{S}$ is defined as $\{1, 2, \ldots, N\}$ where 1 denotes the first port and N the last port to be visited. The set of rows $\mathcal{R}$ is defined as $\{1, 2, \ldots, R\}$ where 1 denotes the bottom row. By $\mathcal{C}$, we denote the set of columns $\{1, 2, \ldots, C\}$.

The binary integer program is based on an upper-diagonal transshipment matrix $T = (T_{ij})$, $i = 1, \ldots, N-1$ and $j = i+1, \ldots, N$. T_{ij} denotes the number of containers to be shipped from port i to port j. Hence, constraint (9.4.2) guarantees that for every port i exactly T_{ij} containers are unloaded in each port j. The constraints (9.4.3) and (9.4.4) imply that there is at most one container stored at each bay position and that the containers are stored on-top of the other. The binary variable y_{ip} denotes whether the bay position p is occupied when leaving port i. The correct shift operation is implied by (9.4.5).

CSP

$$\min \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \sum_{v=i+1}^{j-1} \sum_{r=1}^{R} \sum_{c=1}^{C} x_{ij}(r,c) \tag{9.4.1}$$

$$\text{s.t.} \quad \sum_{v=i+1}^{j} \sum_{r=1}^{R} \sum_{c=1}^{C} x_{ijv}(r,c) - \sum_{k=1}^{i-1} \sum_{r=1}^{R} \sum_{c=1}^{C} x_{kji}(r,c) = T_{ij} \quad \text{for all } i \in \mathcal{S} \setminus \{N\}, \; j \in \mathcal{S} : j > i \tag{9.4.2}$$

$$\sum_{k=1}^{i} \sum_{j=i+1}^{N} \sum_{v=i+1}^{j} x_{kjv}(r,c) = y_i(r,c) \quad \text{for all } i \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{R} \tag{9.4.3}$$

$$y_i(r,c) - y_i(r+1,c) \geq 0 \quad \text{for all } i \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{R} \setminus \{R\} \tag{9.4.4}$$

$$\sum_{i=1}^{j-1} \sum_{l=j}^{N} x_{ilj}(r,c) + \sum_{i=1}^{j-1} \sum_{l=j+1}^{N} \sum_{v=j+1}^{l} x_{ilv}(r+1,c) \leq 1 \quad \text{for all } j \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{R} \setminus \{R\} \tag{9.4.5}$$

$$x_{ijvp}, y_{ip} \in \{0,1\} \quad \text{for all } i,j,v \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{R} \tag{9.4.6}$$

The shift operation is different from the shunting operation in storage yards for trams. This is motivated by the amount of buffer space for temporarily storing the containers to be reloaded onto the ship. The shift operation only counts the number of containers to be removed from the column in order to discharge the containers destinated to the port which the ship is currently visiting. It is assumed that the containers are loaded onto the ship in the order implied by their destination ports, i.e., the container that has to be discharged as last is loaded at first. The amount of rearrangements necessary to achieve the corresponding loading order is neglected. Usually, these temporarily discharged containers are stored next to the quay cranes.

Avriel et al. [APSW98] observed that solving the binary integer program (CSP) becomes impractical even for small instances. However, they introduce a sophisticated heuristic, called the **suspensory heuristic** which is shown to yield good solutions for randomly generated transshipment matrices of different kind.

In [APS96], the container stowage problem is shown to be $\mathcal{NP}$-hard. Avriel et al. present a reduction from the problem of coloring overlap (circular arc) graphs [Ung88] to the container stowage problem. For one column (stack), a

polynomial-time algorithm is presented by Aslidis [Asl89].

9.4.1 Connections between the Container Stowage Problem and the Tram Dispatch Problem

We make use of the binary linear program for the tram dispatch problem in the following way. As we have stated above, the shift operation and the shunting operation differ. If we are only interested in the number of trams that have to be shunted, we can apply the CSP model.

Given a TDP instance, we construct an instance of CSP as follows. For each arrival $a_i \in \mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, we introduce a port i. For each departure $d_j \in \{d_1, d_2, \ldots, d_N\}$, we introduce a port $j+N$. Consequently, the set $\mathcal{S}$ consists of $2N$ ports $1, 2, \ldots, 2N$ where the first N ports correspond to $\mathcal{A}$ and the last N correspond to $\mathcal{D}$.

If we examine the binary linear program of CSP more intensively, we observe that it can be extended to stacks (columns) of different lengths. For each stack $\mathcal{P}_c$, we introduce a column c with P_c positions, $1 \leq c \leq R$ where R denotes the number of stacks in the TDP instance. Hence, the number of positions is defined as $P = \sum_{c=1}^{C} P_c$.

The transshipment matrix $T = (T_{ij})$ denotes how many containers have to be transported from port i to port j. For the tram dispatch problem, T_{ij} indicates if the arriving tram a_i can be assigned to departure d_{j-N}.

For all $a_i \in \mathcal{A}$ and all $d_j \in \mathcal{D}$, we define T as follows. If tram a_i has the same type as departure d_j, we define $T_{i,j+N} = 1$. Otherwise, $T_{i,j+N} = 0$.

In the TDP instance there are usually several trams (and several departures) having the same type. In this case, the constraint (9.4.2) is incorrect for TDP, because each tram a_i has to serve all departures d_j with $T_{i,j+N} = 1$. We can model the matching property of the assignment of trams to departures of the same type by introducing the binary variable s_{ij}. This variable s_{ij} is set to one if and only if departure d_{j-N} is served by tram a_i. In order to modify the CSP model to satisfy this property, we have to add the two following constraints:

$$\sum_{i=1}^{j-1} T_{ij} s_{ij} = 1 \text{ for all } j \in \mathcal{S} \tag{9.4.7}$$

$$\sum_{j=i+1}^{2N} T_{ij} s_{ij} = 1 \text{ for all } i \in \mathcal{S} \tag{9.4.8}$$

Using these notations, we can solve TDP instances by solving the constructed CSP. However, solving the corresponding CSP does not lead to an improvement of the computation times needed to compute optimal solutions for TDP. In fact, solving the corresponding CSP requires significantly more computation time.

TDP–CSP

$$\min \sum_{i=1}^{2N-1} \sum_{j=i+1}^{2N} \sum_{v=i+1}^{j-1} \sum_{c=1}^{C} \sum_{r=1}^{P_c} x_{ij}(r,c) \tag{9.4.1}$$

$$\text{s.t.} \sum_{v=i+1}^{j} \sum_{c=1}^{C} \sum_{r=1}^{P_c} x_{ijv}(r,c) - \sum_{k=1}^{i-1} \sum_{c=1}^{C} \sum_{r=1}^{P_c} x_{kji}(r,c) = T_{ij} \quad \text{for all } i \in \mathcal{S} \setminus \{2N\}, \; j \in \mathcal{S} : j > i \tag{9.4.2}$$

$$\sum_{i=1}^{j-1} T_{ij} s_{ij} = 1 \quad \text{for all } j \in \mathcal{S} \tag{9.4.7}$$

$$\sum_{j=i+1}^{2N} T_{ij} s_{ij} = 1 \quad \text{for all } i \in \mathcal{S} \tag{9.4.8}$$

$$\sum_{k=1}^{i} \sum_{j=i+1}^{2N} \sum_{v=i+1}^{j} x_{kjv}(r,c) = y_i(r,c) \quad \text{for all } \; i \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{P}_c \tag{9.4.3}$$

$$y_i(r,c) - y_i(r+1,c) \geq 0 \quad \text{for all } i \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{P}_c \setminus \{P_c\} \tag{9.4.4}$$

$$\sum_{i=1}^{j-1} \sum_{l=j}^{2N} x_{ilj}(r,c) + \sum_{i=1}^{j-1} \sum_{l=j+1}^{2N} \sum_{v=j+1}^{l} x_{ilv}(r+1,c) \leq 1 \quad \text{for all } j \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{P}_c \setminus \{P_c\} \tag{9.4.5}$$

$$x_{ijvp}, y_{ip} \in \{0,1\} \quad \text{for all } i,j,v \in \mathcal{S}, c \in \mathcal{C}, \; r \in \mathcal{P}_c \tag{9.4.6}$$

The TDP–CSP model can also be applied to the variation of the tram dispatch problem where trams are allowed to leave the depot before the last tram has arrived. Such a problem occurs at railway terminus stations (cf. Section 2.5). In order to apply the model to this situation, we have to define the ports which correspond to the arrivals and departures. We assume that the arrivals and departures are sorted by the times at which they are scheduled. In this arrival-departure sequence, the arrival a_i takes place after departure d_j if and only if $i > j$. An example of such a sequence is given by

$$(a_1, a_2, a_3, d_4, a_5, a_6, d_7, d_8, a_9, d_{10}, d_{11}, d_{12}).$$

Consequently, $\mathcal{A} = \{a_1, a_2, a_3, a_5, a_6, a_9\}$ and $\mathcal{D} = \{d_4, d_7, d_8, d_{10}, d_{11}, d_{12}\}$.

Using these notations, the TDP–CSP model can directly be applied to the terminus dispatch problem. The basic CSP model already fits to the situation that containers are loaded and discharged in every port $2, \ldots, N$. Our extension does not change this property. The constraints (9.4.7) and (9.4.8) require that, in every port corresponding to a departure in $\mathcal{D}$, there have to be more trams than departures of the same type. Otherwise, (9.4.7) and (9.4.8) cannot be satisfied.

As we have already stated for the application of the TDP–CSP model, the computation times increase drastically with the number of vehicles (trams or locomotives) that have to be dispatched in the depot. In particular, if the stacks contain five or more positions, it is impossible to solve arbitrary instances of more than fifteen vehicles within reasonable time. However, the heuristic methods for CSP may be adapted and applied to these situations.

Chapter 10

Container Logistics

In this chapter, we consider a particular dispatching problem arising in maritime container terminals at the example of the "Burchardkai" terminal in Hamburg, Germany. In maritime container terminals, a large number of containers are handled day by day. The containers arrive at the terminal by truck, ship, or train. Before they leave the terminal, they are usually stored in a terminal area, called **yard**. The yard is partitioned into several rectangular areas, called **blocks**. In the blocks, the containers are stored in stacks which are arranged one beside the other and in several rows. In some container terminals, the transport of containers to their storage positions in the yard and to the points at which they leave the terminal is done by manually driven transport vehicles, for instance by **straddle carriers**. In other terminals, like ECT Rotterdam, the transport is carried out by automated guided vehicles. A typical layout of a maritime container terminal is given in Figure 10.0.1. In this thesis, we focus on container terminals in which the containers are carried by straddle carriers.

In maritime container terminals, combinatorial optimization problems arise for instance when assigning vessels to berths, planning the tours for each transport vehicle, or computing good storage positions for the containers. The berth planning problem has been examined by Lim [Lim98] who models this problem as a rectangular packing problem with side constraints. Lim presents a heuristic which is based on (heuristically) computing a longest path in a constructed graph. An alternative network flow approach is due to Chen [CH]. Different versions of tour planning problems for straddle carriers have been considered by Steenken et al. [Ste92a, Ste92b, SHFV93]. The dispatch of straddle carriers for unloading and loading trucks has been modeled as a linear assignment problem which is solved iteratively in real-time [Ste92a, Ste92b]. The combination of different hinterland operations is considered as a traveling salesman problem which is solved heuristically [SHFV93].

In this chapter, we concentrate on the following problem arising in maritime container terminals. We denote this problem as the **combined stowage and transport problem**. The containers destined for one container vessel must be

transported to the vessel before they can be loaded by the quay cranes. For each container, a certain loading position is specified in accordance with the stowage plan. This stowage plan has to be computed on the basis of the information provided by the shipping company. For each bay position, the shipping company indicates the properties that a container must satisfy if it is stored at this position. For instance, the shipping company specifies for a bay position the discharge port, the container type, and its weight. Moreover, restrictions on the stored goods may be given.

Today, the transport of containers to the quay cranes is not taken into account while deciding where each container shall be stored in the bays of the container vessel. The ship planning process starts two days before the vessel arrives at the terminal. At that time, the dispatcher who is responsible for preparing a stowage plan is provided with the actual storage situation of the vessel when leaving the port the vessel has called before. Additionally, the shipping company transmits a preliminary list of containers that shall be loaded. Based on this information, the stowage plan is constructed without considering the transport times for the containers. In this chapter, we present a model for an integrated stowage and transport planning.

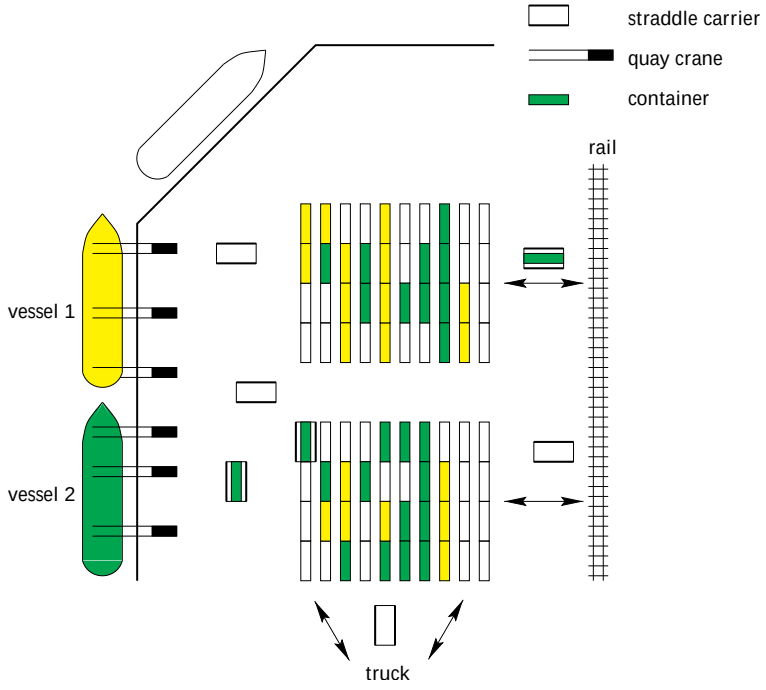

Figure 10.0.1: Container transport in maritime container terminals.

10.1 Ship Planning in Container Terminals

Modern maritime container terminals form an important link in the transport chain of containers. The import and export containers are temporarily stored in the terminal area before they are shipped to their destinations. We consider a subproblem of the corresponding logistic process at the example of a German container terminal.

The turnover at the terminal "Burchardkai" operated by the Hamburg Port and Warehouse Company (HHLA) increased from 1.1 million container units (TEU) in 1992 to 1.6 million TEU in 1998. It is expected that the number of units handled in 2000 will increase to 2.2 million TEU. This large number of container units has to result in improved intelligent logistics in order to deal with this expansion.

At "Burchardkai", more than 3200 vessel calls are operated per year. The loading and discharge process is carried out by quay cranes whereas the transport is performed by a fleet of straddle carriers. The complete dispatch process results in about 10000 container movements per day.

As one part of the logistic process, the ship planning is of high importance for the productivity of a container terminal. The ship planning process is based on a stowage plan provided by the shipping company. A first plan is submitted two days before the arrival of the container ship. This plan consists of a map of the actual stowage situation when the container ship has left the previous port. At this time, it becomes known which containers have to be discharged. Before the arrival, the stowage plan is updated in successive steps with information about the containers to be loaded. Usually, for each bay position a stowage plan fixes where to load a container of a certain weight and bound for a specified discharge port.

Based on this information, the dispatcher determines a stowage plan which fixes an export container for each loading position. After the discharge process, these containers are transported to the quay cranes by straddle carriers. For each bay, the chosen loading strategy implies a linear sequence of containers to be loaded by the crane. The corresponding containers have to be transported from their yard positions to the quay cranes. The transport sequences of the straddle carriers have to meet the loading sequence of the respective crane in order to avoid waiting times during the loading process.

Since a huge number of containers arrive at the terminal by truck, for these containers the concrete delivery times are unknown. Hence, the complete dispatch process suffers from incomplete information so that we have to deal with an online and real-time problem.

10.2 Stowage Planning on Container Vessels

In Section 9.4, we introduced a model for determining a stowage plan for a container ship that visits several ports. This stowage plan specifies for each bay position the destination port for the container which has to be loaded at the position in the given port. In this model, the weights of the containers are not taken into consideration. However, for stability reasons, the container weights must be considered. The containers should be loaded onto the ship in such a way that the heavy containers are stored below the containers having less weight.

A possible stowage plan for one bay is presented in Figure 10.2.2. For each bay position, a **container type** is specified. Hence, at this position, a container having a certain weight and bound for the specified port must be stored. The required size of the container is usually given by the type of the bay. Usually, a bay is restricted to 20' containers or 40' containers. Some bays may contain both types of containers where all the 20' containers are standing on top of the 40' containers.

In the combined stowage and transport problem, the **container type** is defined by

- the container's discharge port,
- the container weight including the weight of the stored goods,
- the type of the container, i.e., its size (20' or 40') as well as special equipment attributes, and
- the kind of goods stored in the container.

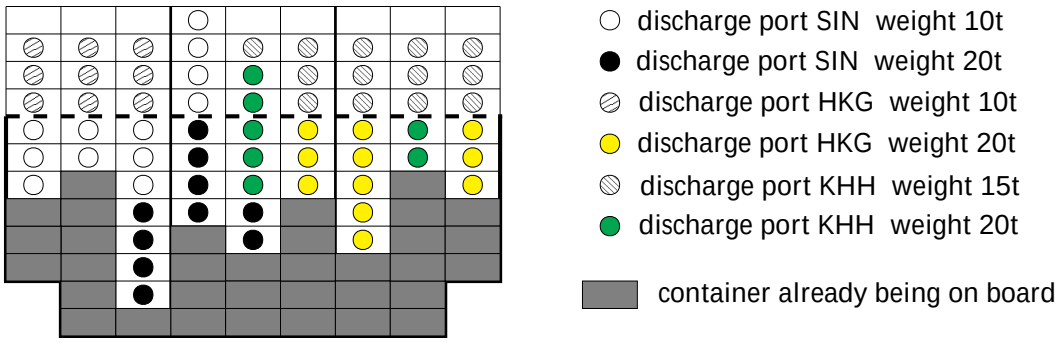

Figure 10.2.2: An example for a stowage plan provided by the shipping company.

Analogously to the tram dispatch problem, we distinguish between the abstract container types described above and the types of containers which may be given by the following list:

General Purpose Container	Hardtop Container
High Cube General Purpose Container	High Cube Hardtop Container
Flat	Open Top Container
High Cube Flat	Platform
Insulated Container	Ventilated Container
Reefer Container	Bulk Container
High Cube Reefer Container	Tank Container

Besides this equipment attributes, the containers may have different lengths, either 20 feet or 40 feet. Some types require that the container is stored at a specially equipped position. For instance, reefer containers have to be kept cool and have to be supplied with electricity. High cube containers differ in height from the standard general purpose containers so that they will probably require two positions.

In the following, we assume that we are given a stowage plan that specifies for each bay position a container type, i.e., the type of the container, its discharge port, and its weight.

10.3 Stowage Planning in Container Terminals

The stowage planning (or ship planning) problem in container terminals differs from the stowage planning problem for container vessels. As discussed in the previous section, for container vessels it suffices to specify a certain container type for each bay position. This (type-based) stowage plan from the shipping company and the list of containers that have to be loaded onto the ship form the basis for the work of the dispatcher at the container terminal. Provided with this information, the dispatcher starts preparing the concrete stowage plan which specifies for each bay position the container to be stored at this particular position. Hence, the dispatcher assigns to each bay position a container whose type matches to the type given for this position.

As mentioned above, a large number of containers to be loaded onto the ship arrive after the loading process has already begun. The dispatcher has to take care about this fact when assigning containers to bay positions. Additionally, the dispatcher must consider the way in which the containers are stored in the stacks of the yard area (cf. Figure 10.3.3).

In order to prevent that a container has to be moved before the container below can be accessed, the terminal company tries to group the containers in such a way that the containers stored in the same stack are of the same type. However, this may not be possible in all situations. In fact, up to 30 percent of the stacks contain containers having different types.

The dispatcher assigns the containers to the bay position in the following way. First, he chooses a bay. Next, he marks the positions at which containers have to be loaded. For these positions, the decision support system gives him

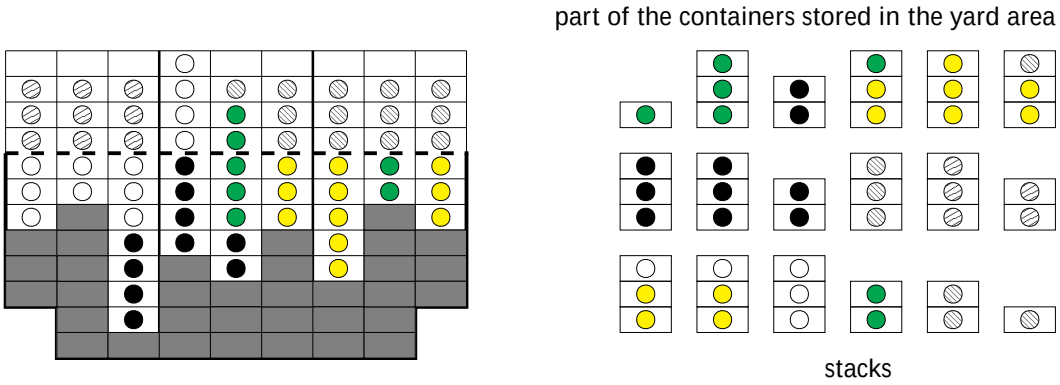

Figure 10.3.3: An example for the storage situation in the yard.

a list of containers that can be assigned. Out of these containers, he chooses those that are assigned to the positions. The concrete assignment is determined by a simple heuristic which assigns the containers in accordance with a specified loading strategy and in accordance with the container weight. After all containers are assigned, the stowage plan is transmitted to the shipping company which accepts the plan or demands some changes.

10.4 Combining Ship and Transport Planning

These days, the stowage plan is computed ignoring the loading and transport sequences. Consequently, the containers are assigned to the bay positions without taking care of the time needed for the transport to the cranes. The containers are stored in stacks of up to three containers in the terminal's yard area.

For each bay position where a container has to be loaded and stored, we assume that the stowage plan provided by the shipping company fixes a certain container type. After fixing a certain loading strategy for each bay, we obtain a linearly ordered sequence of container types to be loaded into the bay (cf. Figure 10.4.4).

Each bay may be partitioned into some partial bays which are considered separately. These partial bays correspond to the bay positions on deck or in the hold of the ship. Moreover, the bay is partitioned into areas that correspond to the hatches. For each partial bay of the vessel, the loading strategy implies a linear order of container types.

The Crane Split

The next decision to be made is the partitioning of the vessel into bay areas that shall be served by one quay crane. This step is called **crane split**. Based on the

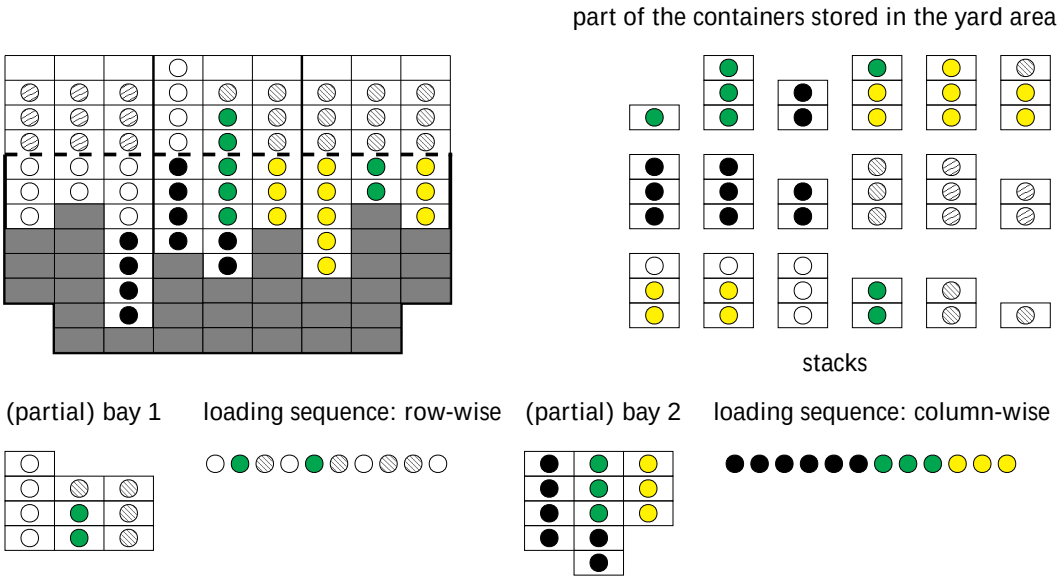

Figure 10.4.4: An example for two loading sequences and loading strategies.

information which crane is accessible at which time, the crane split can be computed by solving a partitioning problem with some operational side constraints. Since the number of cranes available for the loading process is small, it is possible to compute the optimal solution of this partitioning problem within acceptable time. The crane split results in a partitioning of the bay area to be served by the quay cranes.

More formally, we are given a set of bays (or partial bays) $\{1, \ldots, B\}$. Each bay i requires a fixed number b_i of containers to be loaded into the bay, $b_i > 0$. This number is given by the stowage plan provided by the shipping company. We assume that the vessel shall be loaded using C quay cranes where q_c denotes the load of quay crane c, $q_c > 0$ for all quay cranes c. This load q_c corresponds to the amount of time in which quay crane c is available.

Hence, we search for a partition $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_C$ of $\{1, \ldots, B\}$ where each $\mathcal{Q}_i$ contains only consecutive bays, i.e., $\mathcal{Q}_i = \{j_i, j_i+1, \ldots, j_i+k_i\}$ for all $1 \leq i \leq C$. This partition shall be chosen in such a way that the cumulative loads of the cranes differ as minimum as possible. We denote by $Q_i = \sum_{j \in \mathcal{Q}_i} b_j$ the number of containers to be loaded into the bays in $\mathcal{Q}_i$. A possible objective is

$$\min \max_{i,j \in \{1,\ldots,C\}} \mid \frac{1}{q_i} Q_i - \frac{1}{q_j} Q_j \mid .$$

This objective function minimizes the maximum difference between the loads for each pair of quay cranes. For C, a value between 3 and 5 is reasonable for practical problems. The number of bays may differ between 20 and 50. Consequently, it is possible to apply a straight-forward enumeration algorithm to this partitioning

problem. A first upper bound can be obtained by computing for each bay the weighted average load

$$\mu_i = \frac{q_i}{\sum_{i=1}^{C} q_i} \sum_{j=1}^{B} b_j.$$

The corresponding "partition" is obtained as follows: $\mathcal{Q}_1 := \{1, 2, \ldots, k_1\}$ where k_1 is chosen as minimal but such a way that $Q_1 \geq \mu_1$. $\mathcal{Q}_2$ is defined by $Q_2 \geq Q_1 + \mu_2$. The remaining partition is constructed analogously except of the last sets which contain only the remaining bays. Better upper bounds may easily be obtained by slightly varying the values of k_i.

The Loading Strategy for each Bay

For each bay, the dispatcher chooses a loading strategy which specifies in which order the containers shall be loaded into the bay. Since the bays consists of stacks, there are two straight-forward strategies that are used in real-world ship planning. The first possibility is to load the bay column-wise and the second possibility is to load the containers layer by layer. For reasons of visibility, the quay cranes always start with the bay positions at the water-side of the vessel. This fixes a loading sequence for both strategies.

After the dispatcher has decided for each bay which loading strategy will be used, for each bay we obtain a fixed loading sequence of bay positions. In combination with the stowage instructions provided by the shipping company, this results in a sequence of container types that have to loaded into the bays.

Combining the Loading Strategies

For each crane and for each bay, we obtain a loading sequence of container types (cf. Figure 10.4.5). This loading sequence has to be served by a fixed number of straddle carriers per crane. These straddle carriers are also called **van carriers (VC)**. Note that the vessel areas may overlap for some quay cranes if we consider partial bays. In this case, two quay cranes share the loading process for these bays. For instance, a quay crane starts with loading the containers into the bay and is moved to another bay in the next shift while a second quay crane continues with the loading process.

In the following, we assume that there are three straddle carriers available for each quay crane. These straddle carriers transport the containers from their standing position in the yard to the quay cranes. Pooling of straddle carriers is not considered but may be applied to the real-time scheduling problem when more straddle carriers are needed at a quay crane in order to keep the loading process within the time bound.

The loading sequence indicates a sequence of container types. For each loading event of the loading sequence, a container having the same type must be

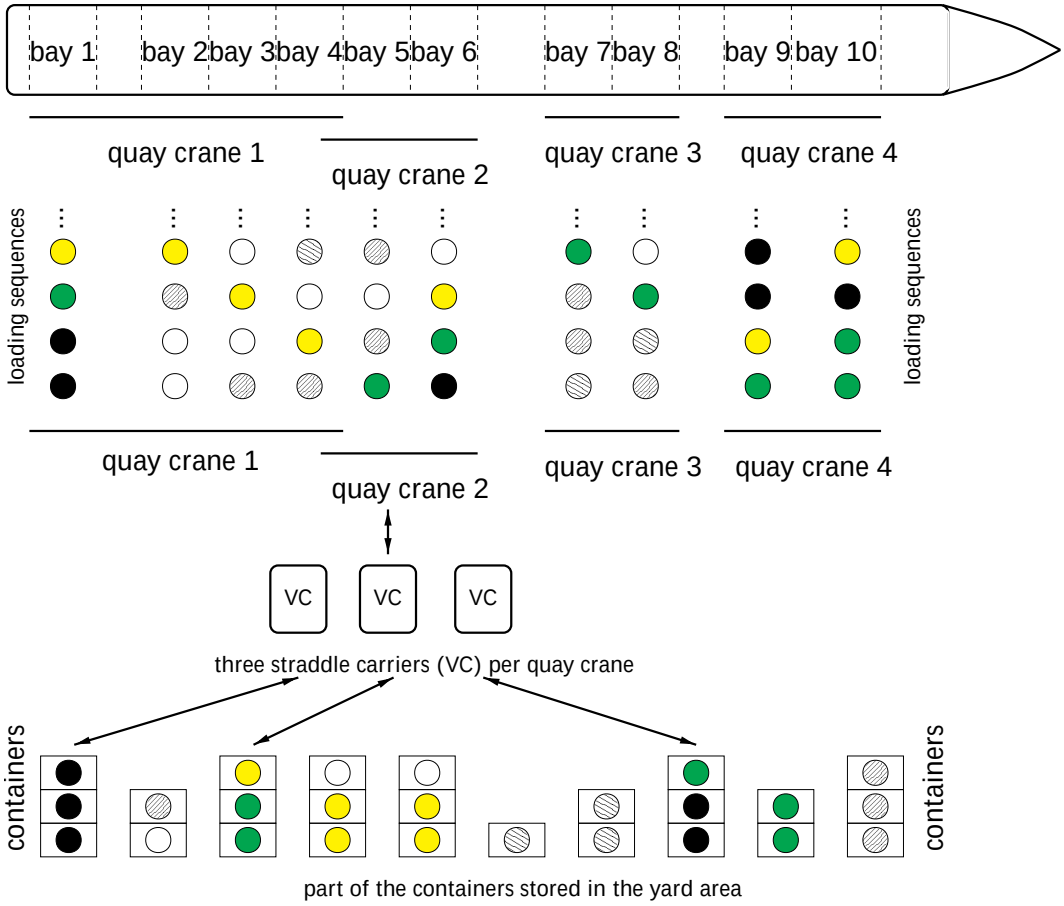

Figure 10.4.5: A possible crane split for four quay cranes and ten bays. To each quay crane, three straddle carriers (VC) are assigned. Using these straddle carriers, all the containers that are to be loaded by the corresponding quay crane have to be transported to the ship. The arrows indicate an assignment of straddle carriers to containers that are to be transported to quay crane 2 and to be loaded into bay 6.

transported to the quay crane at which the containers have to arrive in the order given by the loading sequence. In the case that a straddle carrier is carrying a container for a subsequent loading event, this straddle carrier may have to wait until all containers that are to be loaded earlier have arrived at the quay crane. Below the quay crane, there is only limited space which can be used as a buffer area. For instance, at most two containers can be stored in this buffer area.

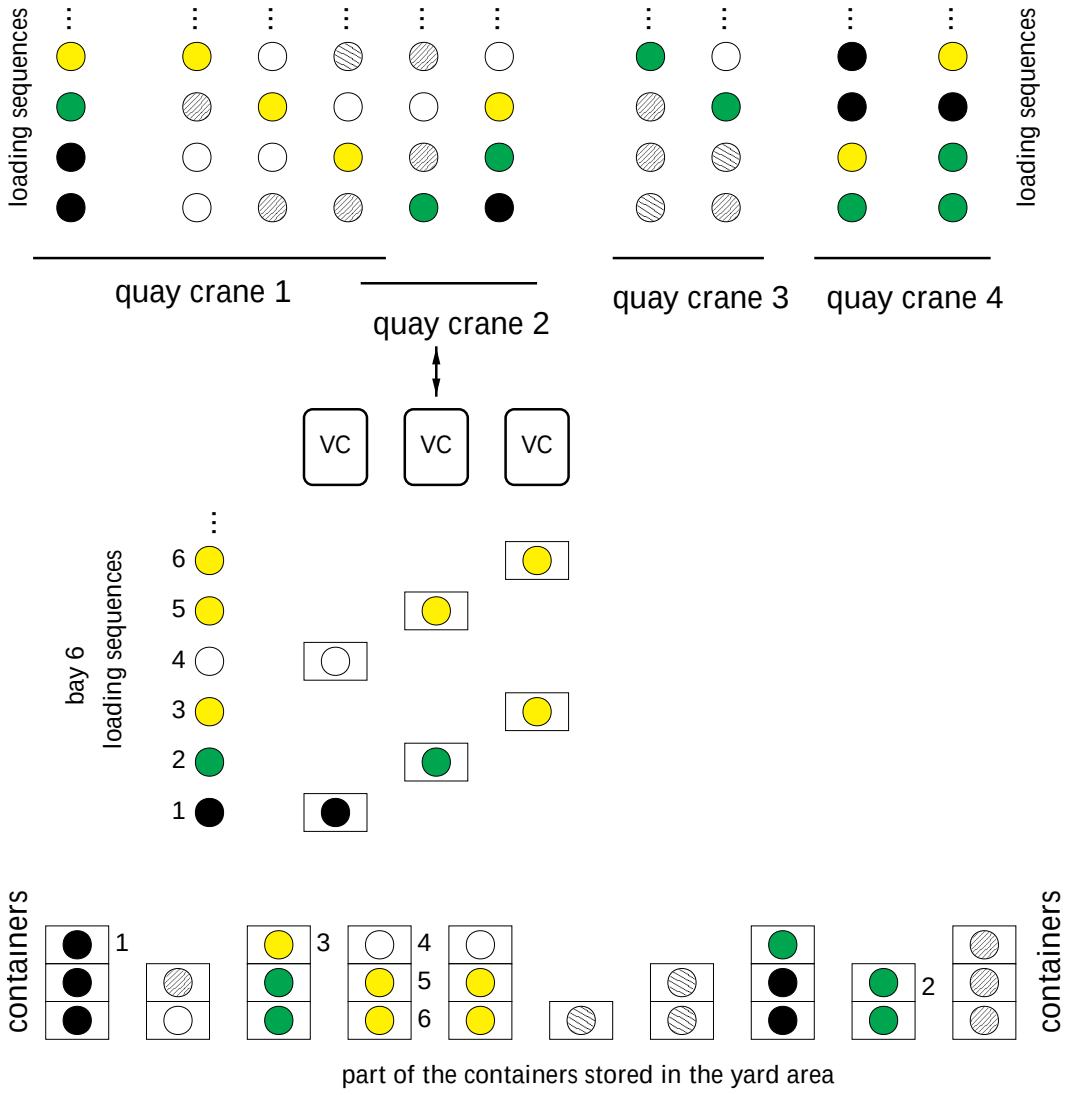

Figure 10.4.6: An example for the assignment of straddle carriers (VC) to the loading events of the loading sequence of bay 6. The containers that shall be loaded are numbered consecutively from 1 to 6 which corresponds to their order in the loading sequence for bay 6. Each straddle carrier has to transport two containers.

The assignment of straddle carriers to transport duties corresponding to the loading events of a loading sequence is shown in Figure 10.4.6. For each loading event i, we choose a container that should be loaded as the i-th container in the loading sequence. This container shall be transported by the specified straddle carrier. Each transport requires a transport time that corresponds to the distance between the stack position of the container and the position of the quay crane.

An example for a **schedule** of straddle carriers is presented in Figure 10.4.7. The transport times are represented by the thick strokes behind the corresponding containers. Note that the second straddle carrier has to start with its first transport some time after the other straddle carriers in order to let the second container arrive at the quay crane after the first container. We observe that for each quay crane we have a scheduling problem with three identical straddle carriers which operate parallel to each other.

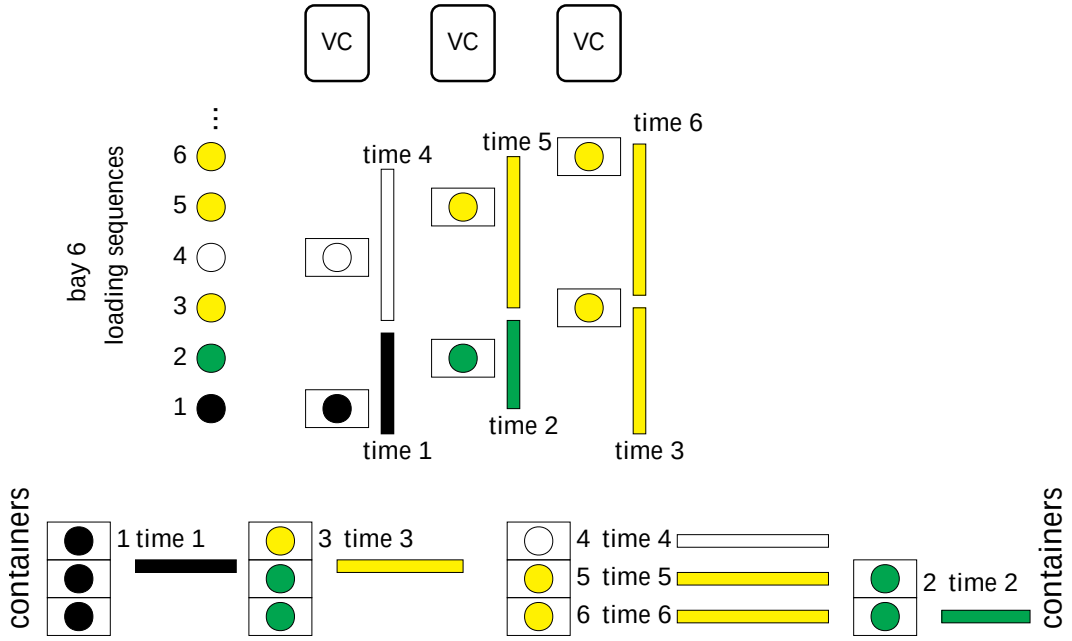

Figure 10.4.7: The scheduling model for the straddle carriers that serve quay crane 2. The container transports must be scheduled in such a way that the containers arrive in the same order as given by the loading sequence of bay 6.

Possible objectives for planning the transport sequences of the straddle carriers are minimizing the time at which the last container is loaded (makespan), minimizing waiting times at the quay cranes during the loading process, or minimizing the lateness of a container transport. We focus on the last objective.

Due to operational constraints, up to 60 boxes (containers) per hour can be loaded onto the ships by the quay cranes. The actual performance of the quay cranes usually differs substantially from the number of boxes that can be loaded

per hour. A loading rate of 30 to 40 boxes per hour would be acceptable. At Burchardkai, during the night shift up to 40 boxes per hour are loaded. During the day, the loading rate is smaller which is caused by different regulations and more congestion in the yard area.

We try to achieve a rate r by planning the loading of the containers at fixed times, for instance every $\frac{3600}{r}$ seconds. As a consequence, every $\frac{3600}{r}$ seconds a container has to be delivered to the bridge. We assume that the first loading event of a container shall be at a time $T > 0$. Then, the second container shall be loaded at time $T + \frac{3600}{r}$, the third container at time $T + 2\ \frac{3600}{r}$ and so on. This results in a parallel scheduling problem with due dates which are process-independent in the sense that we have a due date for the transport of a container of a certain type. The transport planning problem results in a just-in-time scheduling problem with due dates and precedence constraints. Note that this problem is $\mathcal{NP}$-hard, since it contains parallel machine scheduling problem with due dates (cf. for instance [Lia99, CP99]).

10.5 Just-in-time Transport of Containers

Just-in-time scheduling problems have been examined for manufacturing facilities by Steiner and Yeomans [SY93] and for a single machine by Liaw [Lia99] and for parallel machines by Chen and Powell [CP99] who assume a large common due date.

We present a mixed integer program for the scheduling problem with one quay crane. We assume that containers shall be delivered to the crane at times T_i, where $1 \leq i \leq N$ and N denotes the number of containers to be loaded, i.e., the length of the loading sequence. The set $\mathcal{L}$ denotes the set of loading events corresponding to the loading sequence. The crane's loading sequence fixes the container type $t(i)$ for each loading event i. The transport time of a container $c \in \mathcal{C}$ from its yard position to the crane is denoted by p_c. The type of a container $c \in \mathcal{C}$ is denoted by $t(c)$. $\mathcal{C}$ denotes the set of containers and $C =\mid \mathcal{C} \mid$ its cardinality. The set of straddle carriers is given by $\mathcal{V} = \{1, 2, \ldots, V\}$. We achieve the following mixed integer program **(CSTP)** for the **combined container stowage and transport problem**:

$$\min \sum_{i \in \mathcal{L}} L_i \tag{10.5.1}$$

$$\text{s.t.} \sum_{i \in \mathcal{L}} \sum_{v \in \mathcal{V}} x_{civ} \leq 1 \qquad \text{for all } c \in \mathcal{C} : t(c) = t(i) \tag{10.5.2}$$

$$\sum_{c \in \mathcal{C}} \sum_{v \in \mathcal{V}} x_{civ} = 1 \qquad \text{for all } i \in \mathcal{L} : t(i) = t(c) \tag{10.5.3}$$

$$\sum_{j \in \mathcal{L}: j \leq i} \sum_{c \in \mathcal{C}} p_c \, x_{cjv} \leq T_i + L_i \quad \text{for all } i \in \mathcal{L}, v \in \mathcal{V} \tag{10.5.4}$$

$$x_{civ} \in \{0, 1\} \qquad \text{for all } c \in \mathcal{C}, i \in \mathcal{L}, t(c) = t(i), v \in \mathcal{V} \tag{10.5.5}$$

$$L_i \geq 0 \qquad \text{for all } i \in \mathcal{L} \tag{10.5.6}$$

The binary variable $x_{civ} = 1$ if and only if container $c \in \mathcal{C}$ is chosen for loading event $i \in \mathcal{L}$ and transported by straddle carrier $v \in \mathcal{V}$. The variable L_i corresponds to the resulting lateness if the container chosen for loading event $i \in \mathcal{L}$ arrives at the quay crane later than the required time $T_i = T + (i-1)\frac{3600}{r}$.

In this mixed integer program, it is possible that a container assigned to loading event j arrives at the quay crane before the container assigned to event $i < j$. We assume that the buffer space at the quay crane offers space for an arbitrary number of containers.

We discuss computational results for real-world data for four vessels provided by HHLA (cf. Table 10.1). For each quay crane, we solve the CSTP for different lengths of the loading sequence $\mathcal{L}$. We observe that we can determine good solutions within a time bound of one minute computation time. Consequently, one possibility for solving the combined stowage and transport problem in an integrated approach works as follows. We solve the CSTP iteratively for each quay crane and for each part of the loading sequence. The number of iterations depends on the length of the partial loading sequence considered in one step.

As an alternative to solving the mixed integer program CSTP, we introduce the following best-fit heuristic for the parallel planning of all cranes.

Container-Best-Fit (CBF) 10.5.1:

> For each quay crane and each loading event, we always choose the straddle carrier and the (unassigned) container c of the corresponding type so that the time distance between T_i and the actual deliver time Θ of c is minimal. We always prefer those containers for which $\Theta \geq T_i$.

■

First computational results for CBF are presented in Table 10.2. We apply CBF to two real-world instances for different values for the loading rate and

instance	$\mid \mathcal{L} \mid$	Constr.	Vars	Nonzeros	Obj.	LB	CPU sec.
1	30	396	3639	49566	1892	1892	21.89s
	40	443	6289	110151	1892	1892	46.84s
	50	519	6899	177495	1910	1563	$> 60s$
	60	587	7236	248274	1892	1383	$> 60s$
	70	636	7579	321900	2182	1265	$> 60s$
	80	676	11849	428505	1892	1198	$> 60s$
2	30	204	1875	22869	0	0	2.16s
	40	246	2554	44526	0	0	2.38s
	50	428	4955	84807	0	0	9.87s
	60	489	8862	164496	0	0	37.89s
	70	551	9229	255231	0	0	19.01s
	80	603	9662	349329	0	0	61.70s
3	30	261	2031	36369	48	0	$> 60s$
	40	309	2683	62277	35	0	$> 60s$
	50	365	3218	91938	39	0	$> 60s$
	60	409	3804	127926	39	0	$> 60s$
	70	459	4408	170481	43	0	$> 60s$
	80	503	4946	218616	265	0	$> 60s$
4	30	279	1968	29712	94	0	$> 60s$
	40	326	2485	52671	92	92	1.47s
	50	528	3539	81861	122	0	$> 60s$
	60	568	7809	148506	92	0	$> 60s$
	70	608	12079	257451	131	0	$> 60s$
	80	702	15821	406857	92	0	$> 60s$

Table 10.1: Computation results for solving CSTP within a one minute time bound applying CPLEX 5.0 MIP Solver on a Pentium II- 350 MHz PC with 256 MByte core memory. Obj. denotes the lateness, which means $\sum L_i$. LB indicates the best lower bound on Obj. computed within the branch-and-bound process.

the (average) speed of the straddle carriers. We observe that a loading rate of about 40 containers per hour results in a reasonable value for the cumulative lateness and for the makespan. Simulation studies where more side constraints are considered promise a reduction of the time needed to load a vessel.

		570 containers		758 containers	
Rate r	VC speed v $[\frac{m}{s}]$	lateness	last event	lateness	last event
45	1.6	3 h 35'	4 h 46'	6 h 18'	6 h 57'
40	1.6	2 h 28'	4 h 53'	4 h 54'	7 h 09'
36	1.6	1 h 46'	5 h 05'	3 h 35'	7 h 21'
33	1.6	1 h 14'	5 h 23'	2 h 24'	7 h 37'
45	1.8	2 h 11'	4 h 20'	4 h 21'	6 h 21'
40	1.8	1 h 54'	4 h 33'	3 h 02'	6 h 33'
36	1.8	57'	4 h 51'	1 h 52'	6 h 50'
33	1.8	33'	5 h 11'	1 h 15'	7 h 05'
45	2.0	1 h 25'	4 h 04'	2 h 52'	5 h 52'
40	2.0	51'	4 h 22'	1 h 42'	6 h 09'
36	2.0	28'	4 h 42'	1 h 07'	6 h 25'
33	2.0	19'	5 h 03'	47'	6 h 49'
45	2.2	54'	3 h 55'	1 h 44'	5 h 32'
40	2.2	27'	4 h 14'	1 h 02'	5 h 48'
36	2.2	17'	4 h 35'	42'	6 h 11'
33	2.2	13'	4 h 57'	18'	6 h 35'

Table 10.2: Computational results for CBF for different values of r and v: Lateness compared with the time of the last loading event (makespan).

10.6 Real-time Ship Planning

Based on the information available before the container ship enters the port, a stowage plan has to be prepared. This stowage plan is sent to the shipping company asking for acceptance. Next, the accepted stowage plan must be carried out. After the vessel has arrived at its berth position, the quay cranes start with discharging the import containers and those containers that shall be restowed. When a quay crane finishes the discharge process, the loading process starts. According to the corresponding loading sequences for the bay which is actually served, the containers are transported to the crane. This transport is based on the transport sequences which are computed in the combined stowage and transport planning. Unfortunately, the calculated transport times often differ substantially from the real transport times that the straddle carriers need. Consequently, the transport sequences must be adapted in real-time to the different online situations. Since the stowage plan is now fixed and should not be changed, the resulting problem is a just-in-time scheduling problem with fixed due dates for each container. One possible approach to solve this problem in real-time works as follows. Whenever the transport time differs in such a way that we should change the transport sequences of the straddle carriers, we recompute the sequences for the next ten to twenty container transports. The complete transport sequences are updated according to the recomputed solution. The update step may be carried out by applying dynamic programming or heuristics like CBF.

Another online effect which has to result in a change of the solution is a change in the quay cranes' availability. A quay crane may have a defect or, which is more likely, is needed to serve another vessel. As a result, the crane split has to be recomputed so that some bays will be assigned to different quay cranes. Since the corresponding partitioning problem can be solved very fast, we observe that the proposed combined stowage and transport planning approach also suits to this situation.

10.7 Conclusion

In this chapter, we have introduced an integrated approach for a combined stowage and transport planning in container terminals. The basic concept of our model is similar to the model for tram dispatch problem. Given a classification of the export containers into classes of types, we have to assign the containers to the stack positions of the bays so that the type requirements provided by the shipping company are satisfied. In contrast to the tram dispatch problem, we do not have a linear list of containers but a set of containers that are stored itself in stacks (resulting in a partial order of containers). Another difference to the tram dispatch problem is that we have to take into consideration how the containers are transported to the bay position. In doing so, we introduced a just-in-time

scheduling model for combining the specifications of the quay cranes as well as the scheduling problem for the straddle carriers.

The research presented in this chapter is an ongoing project. The first results presented in this thesis show that it is possible to model the stowage and transport problem in maritime container terminals using combinatorial optimization models. Moreover, we have shown that we can apply our methods developed in this chapter in order to solve the considered problem in real-time reacting on different online effects. The future will show if it is possible to improve the real-time planning process in container terminals using the models and methods introduced in this chapter.

Chapter 11

Conclusions

In this thesis, we have considered different dispatching problems which arise in depots of public transport companies and in maritime container terminals. For both problems, we have introduced combinatorial optimization models and methods that could be applied to the *offline* version of these problems as well as to different *online* settings and to real-time scenarios. The dispatching problems were modelled as *stack sorting problems*.

Tram Dispatch

In the tram dispatch problem, arriving trams had to be assigned to stack positions (corresponding to the standing locations in the depot) and to departures (corresponding to the round trips of the next schedule period). The trams serving the round trips of the next schedule period were assumed to leave the depot for the first departure after the last tram serving a round trip of the previous schedule period has arrived. The case that there were departures between some arrivals was considered in connection with the stowage planning problem on container ships (cf. Section 9.4).

Applying the *concept of types*, we modelled the possible assignment of trams to departures. These abstract tram types indicated only feasible assignments of trams to departures and were not necessarily forced to be identical to the trams' car types.

We have considered two different objective functions: minimizing the number of shunting movements and minimizing the number of type mismatches. These problems were shown to be $\mathcal{NP}$-hard. For both problems, we derived mathematical programming formulations. For the problem of minimizing the number of shunting movements, we gave a binary quadratic programming formulation which was a combination of two 0-1-quadratic assignment problems. For the problem of minimizing the number of type mismatches, we introduced a binary linear program. Solving these problems to optimality turned out to require large computation times even for small random instances of 20 trams and real-world

instances of 30 trams. The introduced *last-in-first-out* heuristic *LIFO* was shown to yield (often optimal) solutions within a few seconds of computation time.

If the dispatcher decides to wait with assigning the trams to departures until the last arriving tram has been assigned to a position, we have to consider the departure dispatch problems which are special cases of the above tram dispatching problems. The resulting problems were shown to be $\mathcal{NP}$-hard except of some special cases. For a fixed number of stacks, 0-DTDP and DTMP were shown to be in $\mathcal{P}$.

	general problem	special cases		
	arbitrary number of types	two types	arbitrary number of types	
	arbitrary stack length		stack length = 2	stack length = 3
TDP	$\mathcal{NP}$-hard	open	open	open
TMP	$\mathcal{NP}$-hard	open	open	open
DTDP	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{P}$ for two types	$\mathcal{NP}$-hard
DTMP	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{P}$ for two types	$\mathcal{NP}$-hard

Table 11.1: Complexity results for the different dispatching problems for storage yards.

For the type mismatch problem at departure, we derived a dynamic programming approach that runs in polynomial-time for a fixed number of stacks. This approach could also be applied to the problem of minimizing the number of shunting movements. Firstly, the approach could be used to test whether a shunting-free and type-preserving solution exists. Secondly, it implied an approximation algorithm for the tram dispatch problem at departure based on the relations between the type mismatch and the shunting problem. Thirdly, it could be modified resulting in an non-polynomial dynamic programming approach for the shunting problem. For the shunting problem, we also gave a binary quadratic programming formulation. As heuristic methods, a *greedy heuristic* and *reactive tabu search* were considered.

Computational results showed that random instances up to 30 trams and most of the real-world instances could be solved by the exact methods. Additionally, the *reactive tabu search* was shown to yield optimal solutions for more than 80 percent of the considered instances.

Online and Real-Time Tram Dispatch

The computational results for the random and real-world instances showed that we often obtained a shunting-free and type-preserving solution. Standard competitive analysis requires that, for such instances, the solution value achieved by online algorithms is bounded from above by a constant independent of the number of trams. For the other instances, the competitive ratio may depend on the number of trams.

For the online versions of the considered tram dispatch problems (except of the type mismatch problem), we showed that for some instances the solution value obtained by arbitrary online algorithms was at least linear in the number of trams whereas an optimal offline algorithm yielded a shunting-free and type-preserving solution. A cruel adversary took advantage of this fact so that no online algorithm was competitive in the ordinary sense. However, the performance for instances for which an optimal offline algorithm yielded shunting-free and type preserving solutions did not differ significantly from the performance for other instances. The invented notion of (c, d)-competitiveness integrated both aspects.

The cruel adversary forced the online algorithm to fail by its first decision. Consequently, we observed that randomization did not help. Against an oblivious adversary, we could improve the competitive ratios by a factor of two. Against adaptive adversaries, no improvement was possible.

From a theoretical point of view, all online algorithms yielded "bad" solutions that differed significantly from the possible optimal offline solutions. For the *GREEDY-DTDP* heuristic, we also observed a bad performance for the considered random and real-world instances.

	Lower Bound		Upper Bound	
	c	d	c	d
TDP	$\frac{N-1}{2}$	$\frac{N}{3}$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$
TMP	2	1	N	N
DTDP	$\frac{N}{2}-1$	$\frac{N}{2}-1$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$
DTMP	$\frac{N}{4}$	$\frac{N}{2}$	N	N

Table 11.2: Competitiveness results for the different dispatching problems in storage yards. The upper bound for the shunting problems TDP and DTDP could slightly be improved depending on the number of stacks.

From a practical point of view, for the tram dispatch problem, we could achieve good solutions by applying the online algorithm *LIFO* although it has been shown to require a linear number of shunting movements for some worst-case

Online algorithms				
	Lower Bound		Upper Bound	
	c	d	c	d
GREEDY-DTDP	$\frac{1}{6}N^2 - \frac{1}{2}N + \frac{1}{3}$	$\frac{1}{6}N^2 - \frac{1}{2}N + \frac{1}{3}$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$
GREEDY-DTMP	$\frac{N}{2}$	$N-2$	N	$N-1$

Table 11.3: Competitiveness results for GREEDY-DTDP and GREEDY-DTMP.

instances providing a shunting-free and type-preserving solution. Additionally, the departure problems could be solved by reactive tabu search and the dynamic programming approach.

For the real-time problems, we observed that it was possible to update given solutions within the required time of less than two to five minutes. Based on a given solution, we reacted on changes in the arrival sequence of trams by solving a related problem restricted to the trams, positions, and departures involved in the change. The resulting solutions were often shunting-free and type-preserving or required only a few type mismatches or shunting movements. Moreover, the solutions could be computed within the required computation time of less than two (or five) minutes.

Ship Planning in Container Terminals

Based on the observations for the tram dispatch problem, we introduced a model for the combined stowage and transport planning in container terminals. In this model, we grouped the containers into classes of different types corresponding to the stowage requirements given by shipping company. We presented a just-in-time scheduling algorithm that assigned the containers to the stowage positions in the vessel. This assignment was based on the transport times needed for the transport of the containers from their yard positions to the quay cranes. After the stowage plan has been determined and fixed, the schedule for the straddle carriers could be adapted to several online influences and recomputed in real-time.

Conclusion

We observed that real-world online dispatching problems can be analyzed and modeled using combinatorial optimization methods. Moreover, combinatorial optimization provides methods which can be applied to real-time situations and which yield good solutions for the dispatching problems considered in this thesis.

Notation

$\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$	the set of arriving trams
$\mathcal{D} = \{d_1, d_2, \ldots, d_M\}$	the set of departures
$\mathcal{P} = \{p_1, p_2, \ldots, p_P\}$	the set of stack positions
$\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_T\}$	the set of types
$\mathcal{P}_r$	a subset of $\mathcal{P}$ which contains the positions in stack r, $1 \leq r \leq R$
r	a stack (also often denoted by the corresponding set of positions $\mathcal{P}_r$)
N	the number of trams
M	the number of departures (in the most cases, we assume that $M = N$)
R	the number of stacks
P	the number of stack positions (in the most cases, we assume that $P = N$)
P_r	the number of positions in stack r (in set $\mathcal{P}_r$), $1 \leq r \leq R$
T	the number of types (of trams and departures)
a_i	arriving tram a_i where i denotes the rank of a_i in the (linear) order of arrivals
d_j	departure d_j where j denotes the rank of d_j in the (linear) order of departures
p_q	position p_q in some stack r, $1 \leq r \leq R$
τ	a type of a tram or a departure
t	a mapping that assigns to each tram and each departure a type of $\mathcal{T}$
$X = (x_{iq})$	a 0-1-matrix corresponding to the assignment of trams to positions
$Y = (y_{jq})$	a 0-1-matrix corresponding to the assignment of departures to positions
π_X	an assignment of trams to positions
π_Y	an assignment of departures to positions

$\mathcal{NP}$	class of decision problems that can be solved in polynomial-time by a non-deterministic turing machine
$\mathcal{P}$	class of decision problems that can be solved by a polynomial-time algorithm
$\mathcal{NPC}$	class of decision problems which are $\mathcal{NP}$-complete
$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}$	the set of integral numbers
TDP	tram dispatch problem: the objective is to minimize the number of shunting movements
TMP	type mismatch problem: the objective is to minimize the number of type mismatches
DTDP	the tram dispatch problem at departure: given an assignment of trams to positions, the objective is to minimize the number of shunting movements
DTMP	the type mismatch problem at departure: given an assignment of trams to positions, the objective is to minimize the number of type mismatches
0-TDP, 0-TMP, 0-DTDP, 0-DTMP	corresponding decision problems where we seek for shunting-free and type-preserving solutions
SM	the number of shunting movements required by a given solution
TM	the number of type mismatches required by a given solution
$\mathcal{S}$	state space of a dynamic programming approach
α_{ij}	0-1-coefficient which indicates whether $i < j$
β_{ql}	0-1-coefficient which indicates whether $q > l$
θ_{ij}	0-1-coefficient which indicates whether $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ have the same type
$\bar{\theta}_{ij}$	0-1-coefficient which indicates whether $a_i \in \mathcal{A}$ and $d_j \in \mathcal{D}$ have different types
$\mathbb{R}$	the set of real numbers
$\mathbb{R}_+$	the set of nonnegative real numbers
$\prod_{r=1}^{R}\{0, 1, \ldots, P_r\}$	the cartesian product of the R sets $\{0, 1, \ldots, P_r\}$
$\mid S \mid$	the cardinality of the set S, i.e. number of elements in S

$\emptyset$	the empty set
$\lfloor x \rfloor$	the lower integer part of the real number x, i.e., the largest integer not larger than x
$\lceil x \rceil$	the round up or upper integer part of the real number x, i.e., the smallest integer not smaller than x
$a \equiv b \bmod c$	a is congruent b modulo c

Bibliography

[AFL+94] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Serving requests with on-line routing. In Erik M. Schmidt and Sven Skyum, editors, *Algorithm Theory—SWAT '94: 4th Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes in Computer Science*, pages 37–48, Aarhus, Denmark, 6–8 July 1994. Springer-Verlag.

[AFL+95] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Competitive algorithms for the on-line traveling salesman. In Selim G. Akl, Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 4th International Workshop*, volume 955 of *Lecture Notes in Computer Science*, pages 206–217, Kingston, Ontario, Canada, 16–18 August 1995. Springer.

[AGB96] Arbeitsgemeinschaft Blickpunkt Straßenbahn AGBS, editor. *Straßenbahnatlas Deutschland 1996*. Arbeitsgemeinschaft Straßenbahn e. V., Berlin, 1996.

[AHU75] A. V. Aho, J. E. Hopcroft, and J. D. Ullmann. *The design and analysis of computer algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley, 1975.

[Alb93a] S. Albers. *The Influence of Lookahead in Competitive On-line Algorithms*. PhD thesis, Univ. Saarbrücken, MPI Informatik Saarbrücken, 1993.

[Alb93b] S. Albers. The influence of lookahead in competitive on-line algorithms. Technical report, MPI Informatik Saarbrücken, August 1993.

[Alb97a] S. Albers. Better bounds for online scheduling. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 130–139, El Paso, Texas, may 1997. ACM.

[Alb97b] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, July 1997.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Networks flows: theory, algorithms, and applications*. Prentice-Hall, 1993.

[AP93] M. Avriel and M. Penn. Exact and approximate solutions of the container ship stowage problem. *Computers and Industrial Engineering*, 25(1-4):271–274, 1993.

[APS96] M. Avriel, M. Penn, and N. Shpirer. Container ship stowage problem: Complexity and applications to coloring of circle graphs. Technical report, Faculty of Industrial Engineering and Management, Technion - Israel Institute of Technology, June 1996.

[APSW98] Mordecai Avriel, Michal Penn, Naomi Shpirer, and Smadar Witteboon. Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research*, 76:55–71, 1998.

[Asc95] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, TU Berlin, 1995. Shaker-Verlag, Aachen.

[Asl89] A. H. Aslidis. *Combinatorial Algorithms for Stacking Problems*. PhD thesis, Massachusetts Institute of Technology, January 1989.

[BBH+98] U. Blasum, M. R. Bussieck, W. Hochstättler, C. Moll, H. H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1998.

[BÇ98] R. E. Burkard and E. Çela. Linear assignment problems and extensions. Technical Report Bericht Nr. 127, TU Graz, May 1998.

[BÇPP98] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis. The quadratic assignment problem. Technical Report 126, Technische Universität Graz, May 1998.

[BDB94] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[BDBK+94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[Bel66] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, June 1998.

[BIRS95] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *J. Comput. System Sci.*, 50(2):244–258, 1995.

[BLS92] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task systems. *J. Assoc. Comput. Mach.*, 39(4):745–763, October 1992.

[BT94] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA J. Comput.*, 6(2):126–140, 1994.

[Bur98] R. E. Burkard. Assignment problems. In G. Mehlhorn, editor, *Fundamentals - Foundations of Computer Science*, number 117 in Schriftenreihe der Östereichischen Computer Gesellschaft, pages 31–48. IFIP, 1998.

[Bus98] M. R. Bussieck. *Optimal Lines in Public Rail Transport*. PhD thesis, TU Braunschweig, 1998.

[But97] G. C. Buttazzo. *Hard Real-Time Computing Systems – Predictable Scheduling Algorithms and Applications*. Kluwer, 1997.

[BVA96] Braunscheiger Verkehrs-AG BVAG. Personal communication, 1996.

[BWZ97] M. R. Bussieck, T. Winter, and U. T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79(1–3):415–444, 1997.

[CB95] D. S. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Discrete Appl. Math.*, 61:105–120, 1995.

[Çel98] E. Çela. *The Quadratic Assignment Problem*. Wiley, 1998.

[CGJ95] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithms for Bin Packing: A Survey*, chapter 2, pages 46–93. Cambridge University Press, 1995.

[CH] Chuen-Yih Chen and Tung-Wei Hsieh. A time-space network model for the berth allocation problem. Presented at the 19 th IFIP TC7 Conference on System Modelling and Optimization, 1999.

[CJ91] J. Csirik and D. S. Johnson. Bounded space on-line bin packing: Best is better than first. volume 2 of *Proceedings of the 2nd annual ACM-SIAM symposium*, pages 309–319. SIAM, January 28-30 1991.

[CKPV91] M. Chrobak, H. Karloff, T. H. Payne, and S. Vishwanathan. New results on server problems. *SIADM*, 4:172–181, 1991.

[CN99] M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23:180–185, 1999.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ann. ACM Sympos. Theory Computing*, pages 151–158, 1971.

[CP99] Zhi-Long Chen and W. B. Powell. A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European J. Oper. Res.*, 116:220–232, 1999.

[DHMR] E. Dahlhaus, P. Horak, M. Miller, and J. F. Ryan. The train marshalling problem. submitted to Discrete Applied Mathematics.

[DL77] S. E. Dreyfus and A. E. Law. *The Art and Theory of Dynamic Programming*, volume 130 of *Mathematics in Science and Engineering*. Academic Press, 1977.

[DM97] X. Deng and S. Mahajan. The cost of derandomization: Computability or competitiveness. *SIAM J. Comput.*, 26(3):786–802, June 1997.

[EI71] S. Even and A. Itai. Queues, stacks and graphs. In *Proc. Internat. Symp. Theory of Machines and Computations*, pages 71–86. Academic Press, 1971.

[FKL+91] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[FKT89] Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybernet.*, 9(2):107–119, 1989.

[FW98] Amos Fiat and Gerhard J. Woeginger. *Online Algorithms - The State of the Art*. Number 1442 in Lecture Notes in Computer Science. Springer, 1998.

[FY83] A. M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete Appl. Math.*, 5:89–98, 1983.

[GI89] D. Gusfield and R. W. Irving. *The stable marriage problem: structure and algorithms*. MIT Press, 1989.

[GJ79] M. R. Garey and D. S. Johnson. *Computer and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GJMP80] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods*, 1(2):217–227, 1980.

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Computer Science and Applied Mathematics. Academic Press, 1980.

[GP79] W. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Appl. Math.*, 27:47–57, 1979.

[Gra66] R. L. Graham. Bounds for certain multi-processing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969.

[GW95] G. Galambos and G. J. Woeginger. On-line bin packing - a restricted survey. *Z. Oper. Res. Ser. A-B*, 42:24–45, 1995.

[Ira98] Sandy Irani. *Competitive Analysis of Paging*, chapter 3. In *Lecture Notes in Computer Science* [FW98], 1998.

[JDU+74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.

[Joh74] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. System Sci.*, 8:272–314, 1974.

[Joh90] David S. Johnson. The NP-completeness column: An ongoing guide: The story so far. *J. Algorithms*, 11(1):144–151, 1990.

[Joh92] David S. Johnson. The NP-completeness column: An ongoing guide: The tale of the second prover. *J. Algorithms*, 13(3):502–524, 1992.

[Kam98] N. Kamin. *On-line Optimization of Order Picking in an Automated Warehouse*. PhD thesis, TU Berlin, 1998. Shaker-Verlag, Aachen.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[KB78] L. Kaufmann and F. Broeckx. An algorithm for the quadratic assignment problem. *European J. Oper. Res.*, 2:204–211, 1978.

[KMRS88] A. R. Karlin, M. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[KMV94] S. Khuller, S. G. Mtchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoret. Comput. Sci.*, 127:255–267, 1994.

[KNI93] Y. Karuno, H. Nagamochi, and T. Ibaraki. Vehicle scheduling on a tree with release times and handling times. In *Proceedings of the 4th International Symposium on Algorithms and Computation ISAAC '93*, volume 762 of *lncs*, pages 486–495. Springer, 1993.

[Knu68] D. E. Knuth. *The Art of Computer Programming – Fundamental Algorithms*, volume 1 of *Computer Science and Information Processing*. Addison Wesley, 1968.

[Knu96] D. E. Knuth. *Stable marriage and its relation to other combinatorial problems: an introduction to the mathematical analysis of algorithms*. CRM Proceedings and Lecture Notes. American Mathematical Society, 1996.

[KP91] B. Kalyanasundaram and K. Pruhs. Online weighted matching. In L. A. McGeoch and D. D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 93–94. AMS/ACM, February 1991.

[KP93] B. Kalyanasundaram and K. Pruhs. On-line weighted matching. *J. Algorithms*, 14(3):478–488, 1993.

[KP94] E. Koutsoupias and C. H. Papdimitriou. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science*, volume 35 of *Foundations of Computer Science*, pages 394–400. IEEE, November 20-22 1994.

[KP98] B. Kalyanasundaram and K. Pruhs. *On-line Network Optimization Problems*, chapter 12. In *Lecture Notes in Computer Science* [FW98], 1998.

[Kuh55] H. W. Kuhn. The Hungarian method for the assignment and transportation problems. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[KVV90] R. M. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. volume 22 of *Proceedings of the Symposium on Theory of Computing*, pages 352–358, 1990.

[Lia99] Ching-Fang Liaw. A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Comput. Oper. Res.*, 26:679–693, 1999.

[Lim98] Andrew Lim. The berth planning problem. *European J. Oper. Res.*, 22:105–110, 1998.

[LL85] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. Assoc. Comput. Mach.*, 32:562–572, 1985.

[LRT96] Light Rail Transit Association LRTA. Light rail and modern tramway, April 1996.

[MMS88] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. volume 20 of *Proceedings of the Symposium on Theory on Computing*, pages 322–333, 1988.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MVB96] Magdeburger Verkehrbetriebe MVB. Personal communication, 1996.

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[PSMK90] H. Psaraftis, M. Solomon, T. Magnanti, and T. Kim. Routing and scheduling on a shoreline with release times. *Management Sci.*, 36(2):212–223, 1990.

[Ree95] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, 1995.

[Ric91] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Appl. Math.*, 34:203–227, 1991.

[Rot81] D. Rotem. Stack sortable permutations. *Discrete Math.*, 33:185–196, 1981.

[RS89] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In G. Ausiello, M. Dezani-Ciancaglinin, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming*, number 372 in Lecture Notes in Computer Science, pages 687–703. 1989.

[SHFV93] Dirk Steenken, Andreas Henning, Stefan Freigang, and Stefan Voß. Routing of straddle carriers at a container terminal with the special aspect of internal moves. *OR Spektrum*, 15(3):167–172, October 1993.

[Smi91] D. K. Smith. *Dynamic Programming: A Practical Introduction*. Mathematics and its applications. Ellis Norwood, 1991.

[SPG+97] R. Séguin, J.-Y. Potvin, M. Gendreau, T. G. Crainic, and P. Marcotte. Real-time decision problems: an operational research perspective. *J. Opl. Res. Soc.*, 48:162–174, 1997.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, February 1985.

[Ste92a] D. Steenken. Integrierte DV-Systeme im Container-Umschlag. *Deutsche Verkehrs Zeitung (DVZ)*, 12, Dezember 1992.

[Ste92b] Dirk Steenken. Fahrwegoptimierung am Containerterminal unter Echtzeitbedingungen. *OR Spektrum*, 14:161–168, 1992.

[SY93] G. Steiner and S. Yeomans. Level schedules for mixed-model, just-in-time processes. *Management Sci.*, 39(6):728–735, 1993.

[Tar85] R. E. Tarjan. Amortized computational complexity. *SIAM J. Algebraic Discrete Methods*, 6:306–317, 1985.

[Tom71] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1:173–194, 1971.

[Ung88] W. Unger. On the k-colouring of circle-graphs. In *5th Annual Symposium on Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 61–72, Bordeaux, France, 11–13 February 1988. Springer.

[Ung92] W. Unger. The complexity of colouring circle graphs (extended abstract). In *9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 389–400, Cachan, France, 13–15 February 1992. Springer.

[Vli92] A. Van Vliet. An improved lower bound for on-line packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.

[Wes93] J. West. Sorting twice through a stack. *Theoret. Comput. Sci.*, 117:303–313, 1993.

[Woe93] G. J. Woeginger. Improved space for bounded-space, on-line bin-packing. *SIAM J. Discrete Math.*, 6(4):575–581, 1993.

[WVG96] Wolfburger Verkehrsgesellschaft WVG. Personal communication, 1996.

[Yao77] A. C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. Proceedings of the 17th Annual Symposium on Foundations of Computer Science, pages 222–227, 1977.

[Zei92] D. Zeilberger. A proof of Julian West's conjecture that the number of two-stack-sortable permutations of length n is $2(3n)!/((n+1)!(2n+1)!)$. *Discrete Math.*, 102:85–93, 1992.

Index